STADIA
HELSINGIN AMMATTIKORKEAKOULU

TEKNIIKAN JA LIIKENTEEN TOIMIALA

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

TYÖPÖYTÄSOVELLUS ADOBE AIR -TEKNIIKALLA

Työn tekijä: Tuomas Haimi
Työn valvoja: Erja Nikunen

Työ hyväksytty: __. __. 2008

Erja Nikunen
yliopettaja

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ALKULAUSE

Kiitän Helsingin ammattikorkeakoulu Stadian tietotekniikka-alan opettajia ja oppilaita.

Helsingissä 21.4.2008

Tuomas Haimi

STADIA
HELSINGIN AMMATTIKORKEAKOULU

INSINÖÖRITYÖN TIIVISTELMÄ

Tekijä: Tuomas Haimi	
Työn nimi: Työpöytäsovellus Adobe Air -tekniikalla	
Päivämäärä: 21.4.2008	Sivumäärä: 64 s. + 7 liitettä
Koulutusohjelma: Tietotekniikka	Suuntautumisvaihtoehto: Ohjelmistotekniikka
Työn valvoja: yliopettaja Erja Nikunen	

Työn kirjallinen osuus aloitettiin kertaamalla lyhyesti web-sovellukset ja niiden ominaisuudet. Tekniikan kehittyminen on mahdollistanut rikkaat Internet-sovellukset, joiden periaate ja toiminta esitettiin, ja lisäksi perehdyttiin rikkaiden Internet-sovellusten vaatimuksiin.

Adobe Air on käyttöjärjestelmäriippumaton ajoympäristö työpöytäsovellusten toteuttamiseen web-tekniikoilla. Air-tekniikkaan tutustuttiin käymällä läpi ajoympäristön tärkeimmät komponentit ja niiden merkitys sekä Air-sovelluksen tietoturvamalli. Air-sovellus voidaan toteuttaa vaihtoehtoisilla sovellustekniikoilla, jotka esitellään lyhyesti ja pohditaan valintaan vaikuttavia ominaisuuksia. Lisäksi esitettiin sovelluskehitykseen tarjolla olevia työkaluja ja Air-sovelluskehityksen ominaispiirteitä.

Työssä toteutettiin Asiakasselain-sovellus, jonka on tarkoitus korvata tulevaisuudessa tällä hetkellä käytössä oleva Asiakaspääte-sovellus. Asiakasselaimen avulla asiakas voi etsiä tuotteita ja lisätietoja tavarataloissa. Air-sovelluksen sovellustekniikaksi valittiin HTML/Ajax-tekniikka. Sovellus hyödyntää tavaratalon varasto-tietokantaa sekä verkkokaupan kuva- ja tuoteinformaatiota.

Työn päätavoitteena oli tutustua Adobe Air -tekniikkaan ja sen tarjoamiin mahdollisuuksiin. Air osoittautui mielenkiintoiseksi ja helpoksi vaihtoehdoksi työpöytäsovelluksen toteuttamisessa perinteisillä web-tekniikoilla.

Avainsanat: Adobe Air, Ajax, Flex, rikas Internet-sovellus

STADIA
HELSINGIN AMMATTIKORKEAKOULU

ABSTRACT

Name: Tuomas Haimi

Title: Adobe Air desktop application

Date: 21.4.2008 Number of pages: 64

Department: Information Technology Study Programme: Software Engineering

Instructor: principal lecturer Erja Nikunen

The study began by reviewing web applications and their basic characteristics. New technologies make rich Internet applications possible. Rich Internet applications were described and crucial aspects of the rich Internet applications were explained.

The main focus of the study was to examine Adobe Air technology. Adobe Air is a cross-operating system runtime for developing desktop applications with web-technologies. Air-technology was examined by reviewing Air components and the security model. It is possible to build an Air desktop application with different web-technologies that were reviewed and guidelines for choosing the right technology were given.

A customer service application that allows searching product information in department store was developed as a part of the study. The application was developed using HTML and Ajax technologies with Adobe Air. The application uses databases of the department store and web store.

The main objective of the study was to review Adobe Air and its benefits for developing desktop application. Adobe Air seems to be a very interesting and easy technology for building and deploying rich desktop applications with traditional web-technologies.

Keywords: Adobe Air, Ajax, Flex, rich Internet-application

SISÄLLYS

1 JOHDANTO

Internet on mullistanut tiedonvälitystä ja ihmisten toimintatapoja. Internet on tarjolla työpaikalla, kaupungilla, kotona, ja yhä useammin se kulkee ihmisten mukana kaikkialle. Internet ja etenkin WWW ovat luoneet lähes rajattomat mahdollisuudet kehittää uusia palveluita ja liikeideoita. Lähihistoriaan liittyy monia esimerkkejä siitä, kuinka lähes tyhjästä kehitetyt ideat ovat johtaneet lähes sadun kaltaisiin menestystarinoihin. Kuka muistaa vielä ajan ilman uutispalveluita, verkkokauppoja tai verkkopankkeja? Kehityksen myötä staattiset WWW-dokumentit ovat muuttuneet dynaamisiksi, joiden sisältöä personoidaan käyttäjästä riippuen. Nykyisin monen verkkopalvelun sisältö on täysin käyttäjien itsensä luomaa ja ylläpitämää. Kehitys ei ole jäänyt vain viihdepalveluiden käyttöön. Useat työelämän sovellukset ovat siirtyneet Internetiin. Uudet tekniset innovaatiot ovat mahdollistaneet interaktiiviset ja entistä näyttävämmät sovellukset. Kehitys ensimmäisestä sovelluksesta rikkaisiin Internet-sovelluksiin on ollut nopeaa. Uusia ja entistä parempia tekniikoita julkaistaan lähes viikoittain ja vielä eilen alan paras tekniikka on tänään auttamattomasti vanhaa.

Insinöörityössäni esittelen Adobe Airin, joka on yksi uusimmista ja paljon ennakkokohua herättäneistä tekniikoista. Se yhdistää web-sovelluksen ja työpöytäsovelluksen ominaisuuksia uudella mielenkiintoisella tavalla. Työn tavoitteena on tutkia Air-tekniikan taustoja ja sen tarjoamia mahdollisuuksia. Lisäksi työn tavoitteena on toteuttaa toimiva Air-sovellus, jossa hyödynnetään kirjallisessa tarkastelussa opittuja asioita. Ennen varsinaiseen aiheeseen syventymistä kerrataan kuitenkin lyhyesti web-sovellusten kehittyminen rikkaiksi Internet-sovelluksiksi. Lopuksi tehdään yhteenveto työn tuloksista.

2 PERINTEISET WEB-SOVELLUKSET

Alunperin WWW-dokumentit tarkoitettiin tutkijoiden tiedonvälityskanavaksi, jossa tutkimustietoa voitiin välittää tehokkaasti ja nopeasti tekstipohjaisena. Dokumenttien välille luodaan hyperlinkkejä, joiden avulla voidaan siirtyä helposti dokumenttista toiseen. Alkuperäiseen tarkoitukseen se olikin lähes täydellinen informaation jakelukanava. Vasta ensimmäisen graafisen Mosaic-selaimen jälkeen, vuonna 1993, WWW:n suosio tiedeyhteisön ulkopuolella alkoi lisääntyä. [1.]

Nopeasti lisääntynyt suosio ja uudet graafiset Internet-selaimet, kuten Netscape Navigator ja Internet Explorer nopeuttivat WWW:n kehitystä. Ihmiset halusivat lisätä sivuille tekstin lisäksi kuvia ja multimediaa. Pian WWW:tä alettiin pohtia ohjelmistorajapintana. Visiona oli ns. *webtop*, jossa tietokoneissa aiemmin toimineet työpöytäsovellukset siirrettäisiin suoritettavaksi Internet-selaimeen ja käsiteltävä tieto voitaisiin tallentaa esimerkiksi Internet-palvelimelle. Kehityksen esteenä olivat kuitenkin hitaat ja kalliit tietoliikenneyhteydet, sekä laitteistojen ja etenkin tallennustilan kalleus. Käytössä olleiden tekniikoiden yhteensopivuusongelmat rajoittivat kehitystä omalta osaltaan. [1, s. 37.]

2.1 Web-sovellus

Yleisesti web-sovelluksella tarkoitetaan WWW:n päällä toimivaa sovellusta, jossa Internet-selaimessa suoritettava web-sovellus ja web-palvelin toimivat yhteistyössä toteuttaen erilaisia tehtäviä ja toimintoja. Web-sovelluksen käyttöliittymä toteutetaan HTML-dokumenttina web-selaimessa ja sovelluksen liiketoimintalogiikka web-palvelimella. Tietovarastona käytetään pääasiassa tietokantoja. Nykyisin web-sovellukset ovat tärkeä osa yritysten tietojärjestelmiä. Web-sovellukset ovat voimakkaassa kasvussa erityisesti kaupanalan, pankki- ja pörssipalveluiden ja julkisen hallinnon palveluissa. [3.]

2.1.1 Sovellustekniikat

Web-sovellus toteutetaan yleensä eri tekniikoiden yhdistelmänä. Tekniikoiden tuntemus ja käyttö on yleistynyt siinä määrin, että niitä voidaan yleisesti kutsua web-sovellustekniikoiksi. Tarkemmassa tarkastelussa tekniikat voidaan jakaa karkeasti kahteen ryhmään: Internet-selaimessa toimivan käyttöliittymän toteuttamiseen tarkoitetut tekniikat sekä palvelintekniikat.

HTML-kieli

HTML-kieli on WWW-dokumenttien julkaisuun tarkoitettu merkintäkieli. HTML-kielellä määritetään WWW-dokumenttin rakenne ja sisältö. Dokumentti voi koostua erilaisista teksti-, lomake-, taulukko- ja kuva-elementeistä. Nykyisin on yleistynyt XML-kieleen perustuva XHTML-merkintäkieli, joka mahdollistaa entistä tarkemman rakenteen, ulkoasun ja sisällön toisistaan erottamisen. [4]

Document Object Model (DOM)

DOM on järjestelmä- ja ohjelmointikieliriippumattoman ohjelmointirajapinta HTML- ja XML-dokumenttien käsittelyyn. DOM-malli määrittelee dokumentin loogisen rakenteen ja mahdollistaa ohjelmien ja skriptikielten muokata ja muuttaa dokumentin elementtien sisältöä, näkyvyyttä, järjestystä ja tyylejä. DOM on The World Wide Web Consortium, W3C:n määrittelemä standardi. [5]

Cascading Style Sheets (CSS)

CSS on monipuolinen HTML- ja XHTML-dokumenttien ulkoasun määrittelykieli, tyyliohje. CSS-kielen spesifikaatiot ovat W3C:n määrittelemiä. CSS-kielellä toteutetut säännöt ehdottavat Internet-selaimelle, kuinka dokumentti voidaan esittää. CSS-tyyliohjeita käytetään luonnollisesti yhdessä XHTML-dokumenttien kanssa, koska XHTML-merkintäkieli ei itse sisällä ulkoasuun vaikuttavia ominaisuuksia. Dokumentin sisältö ja rakenne voidaan helposti erotella toisistaan ja samaa tyyliohjetta voidaan käyttää myös muissa dokumenteissa. Tyyliohjeita käyttämällä ulkoasun ylläpito ja muokkaus helpottuu, kun muutokset voidaan kohdistaa ainoastaan yhteen tiedostoon. [6]

JavaScript

JavaScript on kevyt yleiskäyttöinen skriptikieli, jonka tunnetuin käyttötapa on toteuttaa erilaisia dynaamisia toiminnallisuuksia web-sovelluksissa. ECMAScript on JavaScriptin standardoitu versio, joka on suunniteltu sovellusriippumattomaksi skriptikieleksi. Standardista vastaa European Computer Manufacturers Assosiation. Sovellusriippumattomuus saavutetaan jakamalla kieli kahteen osaan: sovellusriippumattomaan runkokieleen ja sovelluskohtaiseen oliomalliin. Web-sovelluksissa käytettävä oliomalli on nimeltään edellä esitetty DOM. [7.]

HTTP-protokolla

HTTP on yleiskäyttöinen, tilaton protokolla, jota käytetään tiedonsiirtoon web-sovelluksen ja -palvelimen välillä. HTTP on pyyntö/vastaus -protokolla, jossa asiakassovellus avaa yhteyden palvelimelle ja lähettää pyynnön. Palvelin vastaa lähettämällä pyyntöä vastaavan vastauksen. Web-sovelluksen toiminta perustuu peräkkäisiin HTTP-siirtotapahtumiin. [8.]

Liitännäiset

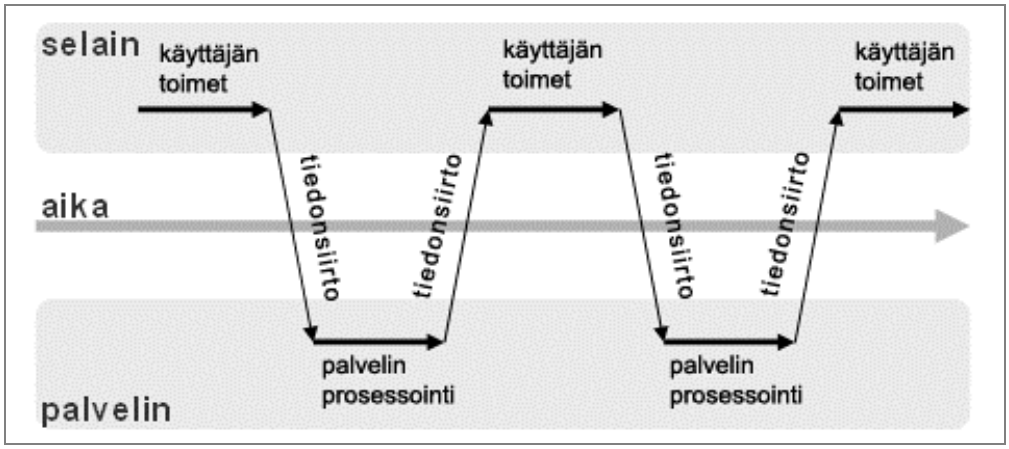
Perustekniikoiden lisäksi web-sovelluksen toteuttamisessa voidaan käyttää erilaisia liitännäisiä, joiden avulla voidaan laajentaa esimerkiksi selaimen tukemia tiedostomuotoja. Yksi yleisimmistä liitännäisistä on Adoben Flash Player, joka mahdollistaa multimedian lisäämisen web-sivuille. Java Appletit perustuvat myös liitännäisiin, mutta niiden merkitys ja käyttö on nykyään vähäistä.

Palvelintekniikat

Palvelintekniikoilla tarjoitetaan tekniikoita, joilla toteutetaan web-sovelluksen palvelinosuus. Tarjolla on monia vaihtoehtoisia tekniikoita, joista mainittakoon tunnetuimpina ASP.net, PHP, JavaEE ja Perl. Lisäksi web-sovellukset käyttävät tietovarastonaan tietokantoja.

2.1.2 Toiminta

Web-sovelluksen toiminta perustuu asiakas-palvelin-malliin, jossa asiakkaan pyynnöt palvelimelle välitetään HTTP-pyynnöillä. Palvelin vastaa asiakkaan lähettämään pyyntöön lähettämällä uuden WWW-dokumentin tai virhetilanteessa virheilmoituksen. Siirtotapahtuma on synkroninen, eli asiakas joutuu odottamaan palvelimen vastausta. Oheisessa kuvassa on esitetty perinteisen web-sovelluksen toiminnan periaate (Kuva 1). [9.]



Kuva 1. Web-sovelluksen toiminta ja synkroninen tiedonsiirto [lähdettä 9 mukaillen]

Kuvaa tulkitsemalla voidaan todeta, että synkroniseen pyyntö-vastaus-malliin perustuvan web-sovelluksen käyttö ei ole tehokasta. Synkronisuus pakottaa käyttäjän odottamaan pyynnön suoritusta ja palvelimen vastausta, ennen kuin voi jatkaa työskentelyä. Useimmiten sovelluksen katkonainen käyttö saa ajatukset pois varsinaisesta työstä. Yksinkertaisissa web-sovelluksissa synkroninen pyyntö-vastaus-malli ei välttämättä haittaa sovelluksen mielekästä käyttöä. Nykyisin web-sovelluksilta vaaditaan kuitenkin paljon enemmän ominaisuuksia ja toimintoja, jotka edellyttävät web-sovellusten uudelleenajattelua. [9.]

2.2 Web-sovellusten edut ja heikkoudet

Web-sovellusten käytön nopeaan lisääntymiseen on vaikuttanut niiden tarjoamat edut. Web-sovellukset sisältävät myös joitakin heikkouksia. Seuraavassa esitetään lyhyesti yleisimpiä web-sovellusten etuja ja heikkouksia.

Edut

Web-sovellukset voidaan toteuttaa käyttöjärjestelmä- ja laitteistoriippumattomasti. Internet-selain löytyy käytännössä kaikista nykyisistä käyttöjärjestelmistä, ja lähes joka paikassa saatavilla oleva Internet-yhteys mahdollistaa web-sovellusten käytön. Päätelaitteeksi kelpaa nykyisin myös monet älypuhelimet ja pda-laitteet. Mobiililaiteiden siirtonopeus, muistikapasiteetti ja prosessointikyky ovat kasvaneet huomattavasti viime vuosina.

Web-sovelluksen versionhallinta on helpommin hallittavissa kuin perinteisten työpöytäsovelluksen. Web-sovelluksen liiketoimintalogiikka sijaitsee web- tai sovelluspalvelimella, johon päivitykset kohdistetaan. Uusin versio on aina kaikkien käyttäjien käytössä. Näin säästetään aikaa, kun sovellusta ei tarvitse päivittää erikseen jokaisen käyttäjän tietokoneelle, ja mahdolliset sovellusvirheet voidaan korjata nopeasti.

Nykyaikainen globaali maailmantalous edellyttää ihmisten mahdollisuutta matkustaa ympäri maailmaa tavatakseen asiakkaita, tavarantoimittajia ja yhteistyökumppaneita. Työntekijällä täytyy olla mahdollisuus päästä matkoillakin hyödyntämään reaaliaikaisesti yrityksen toiminnanohjaus- ja myyntijärjestelmää, sähköpostia tai muita vastaavia sovelluksia. Kun yrityksen sovellukset ovat sijoitettu keskitetysti sovelluspalvelimille, voi työntekijä käyttää sovelluksia periaatteessa mistäpäin maailmaa tahansa. Hänelle riittää tietokone tai nykyaikainen mobiilipäätelaite, jolla voi olla yhteydessä Internettiin.

Sovelluksen ja kaiken käsiteltävän datan ollessa yrityksen palvelimella voidaan niiden tietoturva ja eheys taata luotettavammin. Jos esimerkiksi työtekijän kannettava tietokone varastetaan, yrityksen tiedot eivät joudu asiattomien käsiin, vaan säilyvät turvallisesti yrityksen palvelimella.

Heikkoudet

Yhteensopivuusongelmat muodostuvat pääsääntöisesti käytettävän päätelaitteen fyysistä ominaisuuksista ja Internet-selainten eroista noudattaa Internet-standardeja. Web-sovellusta toteutettaessa on tärkeää testata sovellusta mahdollisimman monella eri laitteisto-, käyttöjärjestelmä ja selainyhdistelmällä. Yhteensopivuusongelmat vähenevät varmasti selainten kehittyessä ja standardien parantuessa. Tähän vaikuttaa etenkin selainmarkkinoiden kiristynyt kilpailu markkinaosuuksista.

Tietoverkon vasteaika tarkoittaa viivettä, joka kuluu käyttäjän pyynnöstä palvelimen vastaukseen. Sovelluksen käytettävyyden yhtenä tärkeänä osana on sen nopeus vastata käyttäjän toimiin. Työpöytäsovellukset vastaavat käyttäjän toimiin välittömästi. Web-sovellus puolestaan hakee tarvittavat tiedot palvelimelta, joka saattaa sijaita maapallon toisella puolella. Lisäksi monimutkaisen pyynnön prosessointiin kuluu aikaa. Web-sovelluksen suunnittelussa on otettava huomioon kuinka suuria määriä dataa on siirrettävä asiakkaan ja palvelimen välillä. Käytettävyyden säilyvyyden kannalta on ilmaistava, että pyyntöön vastaaminen kestää hetken.

3 RIKKAAT INTERNET-SOVELLUKSET

Web-sovellukset ovat muuttuneet entistä monimutkaisemmiksi ja ovat alkaneet kärsiä esimerkiksi käytettävyysongelmista. Perinteisen HTML-pohjaisen web-sovelluksen tekniset rajat on saavutettu ja sovellusten toteuttamiseen tarvitaan uudenlaista ajattelua ja kehittyneempiä tekniikoita. Uudella ajatusmallilla ja tekniikoilla toteutettuja sovelluksia on alettu kutsua rikkaiksi Internet-sovelluksiksi. Konseptin ja sovellusten vaatimukset esitteli ensimmäisenä Macromedia vuonna 2002. Rikkaat Internet-sovellukset pyrkivät parantamaan sovelluksen käyttökokemusta ja tarjoavat perinteistä web-sovellusta kehittyneemmän käyttöliittymän.

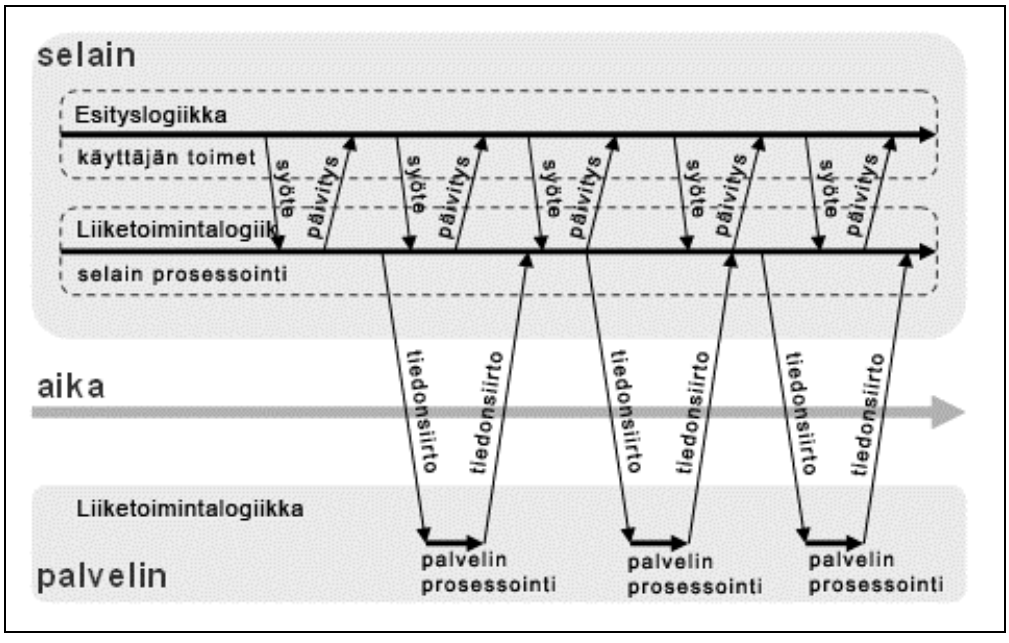
3.1 Sovellusten rakenne

3.1.1 Käyttöliittymä

Rikkaan Internet-sovelluksen käyttöliittymä muistuttaa ulkoasultaan ja toiminnaltaan työpöytäsovellusta. Käyttöliittymät sisältävät monipuolisia käyttöliittymäkomponentteja, ja komponentit toimivat kuin työpöytäsovelluksissa käytettävät vastaavat komponentit. Rikkaat Internet-sovellukset mahdollistavat monipuolisten ja vaativien tehtävien sekä toimintojen suorittamisen. Rikkaat Internet-sovellukset vähentävät asiakkaan ja palvelimen välistä tiedonsiirtoa siirtämällä esityslogiikan ja osan tai koko liiketoimintalogiikan asiakkaan toteutettavaksi. Sovellus kommunikoi palvelimen kanssa ainoastaan silloin, kun se tarvitsee lisää tietoa tai lähettää sitä palvelimelle. Sovellus koostuu yleensä palvelinsovelluksesta, tietokannasta ja palvelimelta ladattavasta asiakassovelluksesta, joka suoritetaan Internet-selaimessa. [10; 11; 12, s. 4 -24]

3.1.2 Asynkroninen tiedonsiirto

Asynkroniset pyynnöt mahdollistavat sovelluksen osittaisen päivittämisen ilman koko sivun uudelleen lataamista (Kuva 2). Palvelin lähettää sovellukseen ainoastaan suorituksen kannalta tarvittavan tiedon. Esimerkiksi verkkokauppasovelluksessa voidaan päivittää ainoastaan ostoskorin sisältö. Sovelluksen käyttö nopeutuu, sillä siirrettävät tietomäärät ovat pieniä. Sovelluksen käyttö on mielekkäämpää ja nopeampaa. [9.]



Kuva 2. Asynkronisen tiedonsiirron periaate [lähdettä 9 mukaillen]

Ongelmat asynkronisessa tiedonsiirrossa

Asynkroninen tiedonsiirto on yksi rikkaiden Internet-sovellusten kulmakivistä, joka lisää sovelluksen käytettävyyttä. Asynkronisuus tuo uusia piirteitä, jotka pitää huomioida etenkin toteutettaessa palvelinsovellusta. Asynkronisten pyyntöjen käyttö vähentää selaimen ja palvelimen välillä siirrettävää tietomäärää, mutta toisaalta sovelluksen lähettämien HTTP-pyyntöjen määrä puolestaan kasvaa. Useat samanaikaiset pyynnöt lisäävät palvelimen kuormaa, joka saattaa aiheuttaa sovelluksen hidastumista. [5] Sovellus voi lähettää useita asynkronisia pyyntöjä palvelimelle samanaikaisesti. Sovelluksen suunnittelussa pitää huomioida tilanne, jossa palvelimelle lähetettävät pyynnöt eivät palaudu samassa järjestyksessä takaisin. Mahdollisena ratkaisuna on pitää yllä tietorakennetta, jolla ylläpidetään samanaikaisten suorituksessa olevien pyyntöjen määrää. Pyyntöjä seuraamalla voidaan rajoittaa sovelluksen samanaikaisten asynkronisten pyyntöjen käynnistämistä ennen edellisten pyyntöjen suoritusten päättymistä.

3.1.3 Sovellustyypit

Rikkaista Internet-sovelluksista voidaan tunnistaa erilaisia sovellustyyppejä, jotka voidaan jakaa neljään ryhmään. Ryhmän jäsenet poikkeavat toisistaan toteutustekniikalta.

Skriptikieleen perustuva web-sovellus

Skriptikieleen perustuvissa rikkaissa Internet-sovelluksissa selaimessa suoritettava sovellus toteutetaan skriptikielellä. Yleisin tällä hetkellä käytössä olevista skriptikielistä on JavaScript. Käyttöliittymä puolestaan toteutetaan HTML-kielellä ja CSS-tyylimäärityksillä, jota muokataan skriptikielellä. XMLHttpRequest-tuen yleistymisen myötä on alettu puhua erityisesti Ajax-sovelluksista. Ajax-termi ja konsepti esiteltiin ensimmäisen kerran Jesse James Garretin vuonna 2005 julkaisemassa artikkelissa: ”Ajax: A new Approach to Web Applications”. XMLHttpRequest-olio mahdollistaa asiakkaan ja palvelimen välisen asynkronisen kommunikaation. Web-sovelluksen käyttöliittymä voidaan toteuttaa yhden web-sivun mallilla, jossa käyttöliittymän eri osia päivitetään tarvittaessa ilman koko sivun uudelleen lataamista. Ajax-sovellukset toimivat hyvin nykyisillä Internet-selaimilla, eivätkä tarvitse erillisiä selaimeen asennettavia liitännäisiä. Ajax-sovellusten saavutettavuus ja käytettävyys on hyvin korkea tasolla. [13.]

Liitännäiseen perustuva web-sovellus

Liitännäisiin perustuvat sovellukset tarvitsevat toimiakseen erillisen Internet-selaimeen asennettavan liitännäisen, joka on yleensä kolmannen osapuolen tarjoama. Liitännäiset tarjoavat monipuoliset mahdollisuudet toteuttaa interaktiivisia ja näyttäviä sovelluksia. Liitännäisiin perustuvia web-sovellustekniikoita on esimerkiksi Adobe Flash ja Flex, jotka toimivat Adobe Flash Playerin -liitännäisen avulla. Lisäksi Laszlo Systemin OpenLaszlo-sovellukset toimivat Flash Playerin avulla. Muita liitännäisiin perustuvia sovellustekniikoita on Java Appletit ja Microsoftin Silverlight-sovellukset. Liitännäisiin perustuvien sovellusten ongelmana on liitännäisen levinneisyys. Ilman selaimeen asennettua liitännäistä sovellusta ei voida käyttää. Liitännäisiin perustuva sovelluksen saavutettavuus on pienempi, kuin skriptaukseen perustuvan sovelluksen. Toisaalta liitännäiset mahdollistavat graafiset ja monipuoliset sovellukset ja ovat yleensä myös suorituskykyisempiä. [13.]

Internet-selaimeen perustuva web-sovellus

Selaimeen perustuvat web-sovellukset tukeutuvat Internet-selaimen sisältämään tukeen tulkata ja renderöidä deklaratiivisesti määriteltyjä käyttöliittymiä. Yleensä näin toteutetut rikkaat Internet-sovellukset toimivat ainoastaan tietyssä Internet-selaimessa. Esimerkkinä Mozillan XUL-merkintäkielellä toteutetut sovellukset. [13; 14.]

Internet-toimitukseen perustuva web-sovellus

Internet-toimitukseen perustuvat web-sovellukset ladataan ja asennetaan Internetistä käyttäjän tietokoneelle. Sovellukset suoritetaan Internet-selaimesta erillään olevassa, rajatussa suoritusympäristössä. Sovellukset hyödyntävät verkkoyhteyttä ja mahdollisesti erillistä palvelinsovellusta. Web-toimitukseen perustuvat sovellukset muistuttavat normaaleja työpöytäsovelluksia, mutta sovellusten pääsy paikallisiin resursseihin voi olla rajattu. Web-toimitukseen perustuvia tekniikoita ovat Java Web Start -sovellukset ja Adobe Air -sovellukset. [13.]

3.2 Sovelluksen vaatimukset

Rikkaille Internet-sovelluksille on määritetty vaatimuksia, joita sovelluksen pitää toteuttaa. Rikkaat Internet-sovellukset yhdistävät työpöytäsovelluksien monipuoliset ja interaktiiviset toiminnot web-sovellusten hyvällä saavutettavuudella. Sovellukselta vaaditaan työpöytäsovellusten tasoista tuottavuutta, jonka puute on yleensä ollut perinteisten web-sovellusten ongelma. Seuraavassa on esitetty rikkaille Internet-sovelluksille asetettuja vaatimuksia suoritusympäristön, käytettävyyden ja käyttöliittymän sekä tiedonhaun osalta.

Suoritusympäristö

Rikkaat Internet-sovellukset suoritetaan yleensä Internet-selaimessa, joka on tarjolla lähes jokaiseen käytössä olevaan käyttöjärjestelmään. Selaimessa suoritettavat sovellukset eivät voi käsitellä paikallisen tietokoneen resursseja. Suoritusympäristönä voi olla myös erikseen käyttäjän tietokoneelle asennettava ajoympäristö. Ajoympäristön ominaisuuksia tarkastellaan tarkemmin luvussa 4.1. [10.]

Käyttöliittymä ja käytettävyys

Sovellusten pitää tarjota samankaltaisia interaktiivisia käyttöliittymäkomponentteja kuten normaalit työpöytäsovellukset. Komponenttien tulisi toimia samankaltaisesti työpöytäsovelluksen vastaavien komponenttien kanssa. Yhdenmukaisuus helpottaa uuden sovelluksen opettelua ja parantaa sovelluksen tuottavuutta. Käytettävyyteen liittyy myös mahdollisuus käyttää sovellusta erityyppisillä päätelaitteilla säilyttäen yhtenäisen käyttökokemuksen. [10.]

Tiedonhaku

Palvelinkommunikointiin käytettävät vasteajat pitäisi olla mahdollisimman lyhyet, ettei sovelluksen käytettävyys kärsi. Asynkroninen kommunikointi on yksi rikkaiden Internet-sovellusten perusominaisuuksista. Sovelluksen pitäisi pystyä kommunikoimaan samanaikaisesti usean eri tietolähteen kanssa. Yhdistelemällä tietoa useasta eri lähteestä, voidaan luoda entistä monipuolisempia sovelluksia. Lisäksi rikkaiden Internet-sovellusten vaatimuksena on säilyttää sovelluksen osittainen toiminta palvelinyhteyden katketessa. [10.]

4 ADOBE AIR

4.1 Mikä on Adobe Air?

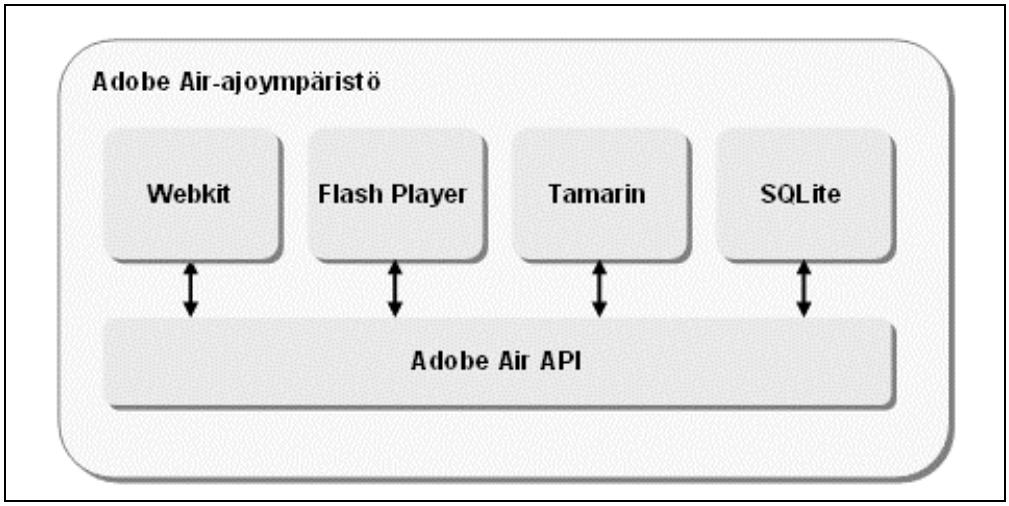
Air on Adoben kehittämä ja julkaisema käyttöjärjestelmäriippumaton ajoympäristö rikkaiden työpöytäsovelluksien toteuttamiseen web-tekniikoilla. Air tulee sanoista *Adobe Integrated Runtime*. Aiemmin Air tunnettiin nimellä Apollo. Tällä hetkellä uusin versio on helmikuussa 2008 julkaistu Abode Air 1.0. Ajoympäristö julkaistiin ensimmäisenä Windows- ja Mac OS X-käyttöjärjestelmille. Tuki Linux-käyttöjärjestelmille on tarkoitus lisätä myöhemmin samana vuonna. Tulevaisuuden suunnitelmissa on myös tuen lisääminen mobiililaitteille. [15.]

Ajoympäristöllä tarkoitetaan yleensä käyttöjärjestelmästä erillistä ja turvallista suoritusympäristöä, joka tarjoaa perustoiminnallisuudet ajoympäristössä suoritettavalla sovellukselle. Toiminnallisuuksia ovat esimerkiksi sovelluksen sisällön renderöinti, verkkotoiminnot sekä käyttäjän ja sovelluksen välinen vuorovaikutus. Sovellus kehitetään suoraan ajoympäristölle, jolloin se toimii täsmälleen samoin kaikissa ajoympäristöä tukevissa käyttöjärjestelmissä. Ajoympäristön tehtävä on huolehtia sovelluksen toimimisesta ja piilottaa käyttöjärjestelmien väliset erot. Tyypillisesti ajoympäristönä toimii Internet-selain, kuten Internet Explorer tai Mozilla Firefox. Erittäin laajalle levinneitä ja suosittuja ajoympäristöjä ovat myös Adobe Flash Player ja Javan virtuaalikone. [16.]

Air-ajoympäristö tuo rikkaat, interaktiiviset ja dynaamiset sovellukset käyttäjän työpöydälle. Sovellus voidaan toteuttaa perinteisillä web-tekniikoilla, kuten HTLM, CSS ja JavaScript tai Adoben Flash- ja Flex-tekniikoilla. Adobe Air tukee myös useita Ajax-kirjastoja ja sovelluskehyksiä. Ajoympäristö asennetaan käyttäjän tietokoneelle ensimmäisen sovelluksen yhteydessä, jonka jälkeen Air-sovellukset käyttäytyvät samoin kuin normaalit työpöytäsovellukset. Air-sovellus yhdistää web-sovelluksen ja työpöytäsovelluksen ominaisuuksia.

4.1.1 Ajoympäristön komponentit

Air-ajoympäristö koostuu neljästä pääkomponentista sekä erillisistä rajapinnoista (Kuva 3). Air-ajoympäristön komponentit perustuvat pääsääntöisesti avoimeen lähdekoodin, paitsi Adoben oma Flash Player. Komponentit on valittu erityisesti pienen koon ja tehokkuuden perusteella, jonka avulla ajoympäristön koko on saatu mahdollisimman pieneksi ja suorituskykyiseksi. Air 1.0 -versiossa ajoympäristön koko on noin 11 megatavua. [15.]



Kuva 3. Ajoympäristön komponentit

Webkit on avoimeen lähdekoodiin perustava HTML-moottori, joka vastaa Air-sovelluksessa HTML-, CSS- ja JavaScript -kielten tulkkauksesta ja tuottamisesta. Webkit tukee seuraavia web-tekniikoita: XHTML, CSS, JavaScript, W3C DOM Level 2 ja XMLHttpRequest. Webkit on käytössä esimerkiksi useissa Applen Mac OS X-ohjelmissa kuten Safari ja Dashboard. Toinen huomattava Webkitin hyödyntäjä on Nokia, joka käyttää Webkit HTML-moottoria S60-sarjan puhelimien Internet-selaimissa. [17.]

Flash Player on Air-ajoympäristön toinen pääkomponentti ja se vastaa Flash- ja Flex-tekniikoilla tuotetun sisällön esittämisestä. Flash mahdollistaa graafisemmat sovellukset. Sen avulla sovelluksen voidaan liittää vektorigrafiikkaa, kuvia, ääntä, videoita ja 3D-animaatiota. Air mahdollistaa HTML- ja JavaScript-kielillä toteutetun sovelluksen hyödyntää Flash Playerin tarjoamia monipuolisia rajapintoja. [17.]

Tamarin on Mozilla Foundationin avoimeen lähdekoodiin perustuva ECMAScript-tulkki. Sen kehityksessä on kiinnitetty erityisesti huomiota suorituskykyyn. Tällä hetkellä Tamarin tukee ECMAScript 3rd -versiota ja osittain

neljännen version määrityksiä. Tamarin tulee jatkossa vastaamaan Firefox-selainen JavaScript-tulkkauksesta. Air-ajoympäristö käyttää Tamarin-tulkkia ActionScriptin tulkkaukseen. [17.]

Adobe Air sisältää sisäänrakennetun relaatiotietokannan, joka perustuu avoimen lähdekoodin SQLite-tietokantaan. SQLite on pieni ja tehokas tietokanta, joka ei vaadi erillistä palvelinta, vaan toimii suoraan kohdekoneessa. Tietokanta tallennetaan suoraan käyttäjän tietokoneelle. SQLite on kirjoitettu C-kielellä, joten se voidaan käytännössä kääntää mille tahansa käyttöjärjestelmälle. SQLite on käytössä monissa eri sovelluksissa. Esimekiksi Google Gears käyttää SQLite-tietokantaa tarjoamaan web-sovelluksille offline-tuen. Adobe Air käyttää SQLite-tietokantaa osittain samaan tarkoitukseen. Lisäksi sen avulla voidaan toteuttaa työpöytäohjelmia, jotka hyödyntävät tietokantaa ilman erillistä tietokantapalvelinta. [17; 18.]

4.1.2 Ajoympäristön rajapinnat

Air-ajoympäristö sisältää pääkomponenttien lisäksi joukon ohjelmointirajapintoja, jotka mahdollistavat monipuolisen työpöytäsovelluksen toteuttamisen. Rajapintoja voidaan hyödyntää Air-sovelluksissa sovellustekniikasta riippumatta. Etenkin HTML- ja JavaScript-tekniikoilla toteutettavissa sovelluksissa rajapinnat tuovat huomattavasti laajemmat mahdollisuudet suunnitella ja toteuttaa interaktiivisia, dynaamisia ja käytettävyydeltään parempia sovelluksia. Seuraavaksi esitetään osa tärkeimmistä ja mielenkiintoisimmista ajoympäristön tarjoamista rajapinnoista.

Window ja System Chrome

Ikkunointirajapinta mahdollistaa natiivien käyttöjärjestelmäikkunoiden hyödyntämisen Air-sovelluksissa, mikä lisää Air-sovelluksen yhtenäisyyttä muiden työpöytäohjelmien kanssa. Rajapinnan avulla ikkunoita voidaan luoda, sulkea ja käsitellä Air-sovelluksesta käsin, sekä reagoida ikkunoiden tuottamiin tapahtumiin. Ikkunat voidaan myös toteuttaa ilman käyttöjärjestelmän sisältämiä kehyksiä. Sovellusikkuna voi olla esimerkiksi läpinäkyvä ja mielivaltaisen muotoinen, joka antaa lähes rajattomat mahdollisuudet vaikuttaa sovelluksen ulkoasuun. [15.]

File I/O ja Byte Array

Tiedostorajapinta mahdollistaa paikallisten tiedostojen ja kansioiden käsittelyn Air-sovelluksesta käsin. Tiedostojen ja kansioiden luonti, poisto, nimeäminen ja kopioiminen voidaan toteuttaa asynkronisesti tai synkronisesti. Asynkroninen tiedostojenkäsittely mahdollistaa raskaat tiedosto-operaatiot sovelluksen taustalla lukitsematta sovelluksen muuta toimintaan. Tiedostorajapinta tarjoaa myös sovelluskohtaisen salatun tiedontallentamismahdollisuuden, jota voidaan hyödyntää esimerkiksi salasanojen ja tunnusten tallennuspaikkana. Salatuille tiedostoille on varattu 10 megatavua tallennustilaa sovellusta kohden. Byte Array -rajapinta mahdollistaa binääri-muodossa olevien tiedostojen lukemisen ja kirjoittamisen. Sen avulla Air-sovelluksessa voidaan esimerkiksi lukea ja kirjoittaa zip-tiedostoja. [15.]

Drag and Drop ja Clipboard

Raahaa ja pudota -rajapinta mahdollistaa Air-sovellusten ja normaalien työpöytäsovelluksien välisen nopean ja helpon tiedostojen vaihdon. Esimerkiksi kuvatiedostot voidaan viedä helposti Air-sovellukseen siirtämällä tiedostot hakemistosta suoraan Air-sovellukseen. Rajapinta mahdollistaa raahaa- ja pudota-toiminnot myös Air-sovelluksen eri komponenttien välillä. Leikepöytärajapinta puolestaan mahdollistaa käyttöjärjestelmän tarjoaman leikepöydän hyödyntämisen Air-sovelluksissa. Sen avulla voidaan tuoda tietoa Air-sovellukseen leikepöydältä ja vastaavasti lisätä tietoa leikepöydälle muiden sovellusten käytettäväksi. Molemmat rajapinnat tukevat tällä hetkellä ainakin kuva-, teksti- ja HTML -tiedostoja sekä URL-linkkejä. Rajapintojen avulla Air-sovellusten käytettävyys parantuu, kun sovellukset toimivat samoin kuin perinteiset työpöytäohjelmistot. [15.]

Database

Tietokanta-rajapinnan avulla Air-sovelluksessa voidaan hyödyntää ajoympäristön sisältämää SQLite-relaatiotietokantaa. Tietokanta ja sen käyttämä tiedosto tallennetaan paikallisesti käyttäjän tietokoneelle, joten tietokannan käyttöön ei tarvita verkkoyhteyttä. Ajoympäristöön sisään rakennettu tietokanta tarjoaa monipuolisten sovellusten toteuttamisen. Paikallista tietokantaa voidaan esimerkiksi käyttää väliaikaisena tallennustilana, kun verkkoyhteys varsinaiseen tietokantapalvelimeen on estynyt. Tietokantaa voidaan käsitellä asynkronisesti, mikä lisää huomattavasti sovelluksen käytettävyyttä. Tietokannan synkroninen käyttö on myös mahdollista. [15.]

Network

Ajoympäristön sisältämä verkko-rajapinta mahdollistaa monipuoliset verkkotoiminnot Air-sovelluksessa. Hyödyllinen ominaisuus on tarkkailla verkkoyhteyden tilaa, jolloin sen tilassa tapahtuviin muutoksiin voidaan helposti reagoida. Air-sovelluksessa voidaan käyttää HTTP-verkkoyhteyden lisäksi Socket-yhteyksiä ja kommunikoida paikallisesti Air-sovelluksien välillä. Ajoympäristö mahdollistaa kommunikoimisen eri domainien välillä, joka on estetty esimerkiksi Internet-selaimissa. [15.]

4.1.3 Air-sovellustekniikat

Adobe Air -työpöytäsovellukset voivat hyödyntää tehokkaasti eri web-palveluita. Sovellus voi tukeutua kokonaan sovelluspalvelimen palveluihin tai se voi toimia täysin itsenäisesti ilman verkkoyhteyttä paikallisena työpöytäsovelluksena. Air-sovellus on tyypiltään web-toimitukseen perustuva sovellus. Sovelluksen toteuttamiseen ei tarvitse opetella uutta ohjelmointikieltä. Sovelluskehityksessä käytetään tunnettuja web-sovellustekniikoita. Air-sovellustekniikat voidaan jakaa kolmeen pääryhmään: HTML/Ajax-, Flash- ja Flex-tekniikoihin.

Ajoympäristö sisältää avoimeen lähdekoodin perustuvan Wekbit HTML-moottorin. Webkit tukee XHTML-, JavaScript-, CSS- ja DOM-tekniikoita, lisäksi Webkit tukee myös XMLHttpRequest-oliota. Air-sovellus voidaan siis toteuttaa Ajax-tekniikalla. HTML- ja JavaScript-koodi tulkataan ja suoritetaan Air-sovelluksessa ajonaikaisesti.

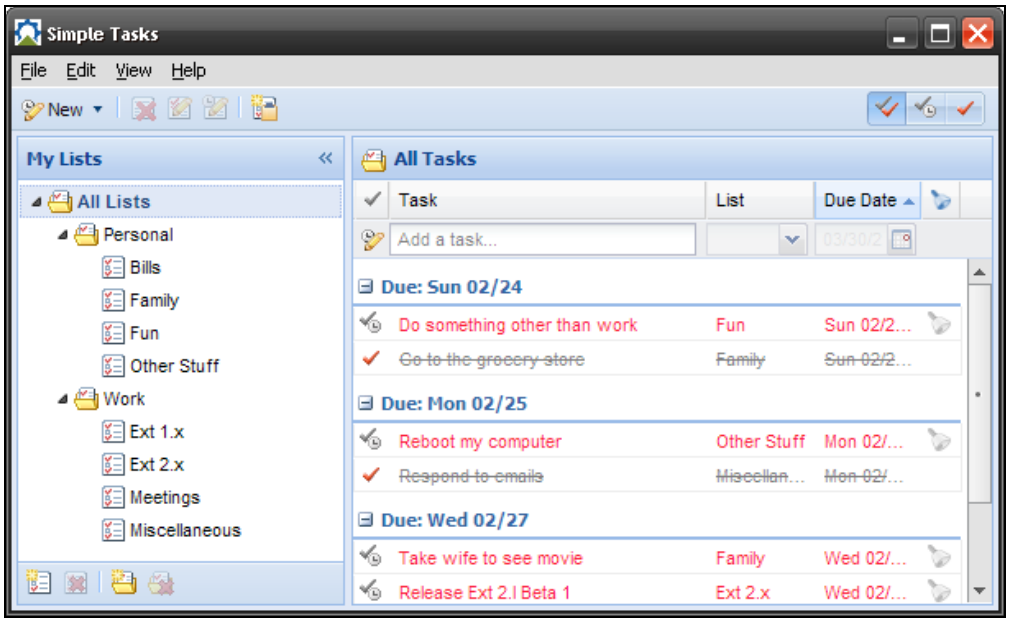
Toinen Air-sovelluskehitykseen tarkoitetuista päätekniikoista on Flex. Adobe Flex on Flashin rinnalle kehitetty ohjelmointiperusteinen tekniikka, joka soveltuu hyvin vuorovaikutteisten ja rikkaiden Internet-sovellusten toteuttamiseen. Sovelluksen käyttöliittymä ohjelmoidaan XML-kieleen perustuvalla MXML-merkkauskielellä ja ohjelmointikielenä on JavaScriptin sukulainen, ActionScript. Flex-sovellukset käännetään binäärisiksi SWF-tiedostoiksi. Flex-sovelluksen suorituskyky on yleensä ajonaikaisesti tulkattavaa HTML-sovellusta parempi. Flex sisältää paljon valmiita käyttöliittymäkomponentteja, joka nopeuttaa sovelluskehitystä. Käytössä on myös erilaisia tehosteita, joiden avulla sovelluksista saadaan todella näyttäviä. Ensimmäinen Flex-versio ei saavuttanut vielä suurta suosiota kalliin hintansa vuoksi. [20, s.1-12; 21.]

Air-sovelluskehitykseen voidaan käyttää myös Adobe Flash -tekniikkaa. Flash mahdollistaa näyttävät graafiset sovellukset, sekä äänen ja videoiden tehokkaan hyödyntämisen sovelluksissa. Air-sovelluksessa voidaan käyttää monipuolisia Flash Playerin tarjoamia rajapintoja. Adobe Flash on lähes standardin asemaan noussut tekniikka näyttävien ja interaktiivisten pelien ja sovellusten toteuttamisessa. Flash perustuu aikajanatekniikkaan, joka saattaa olla ohjelmoijille vieras tapa toteuttaa sovellus. Flash-sovellusten toiminnallisuus perustuu ActionScriptiin. Flash-tekniikkaa voidaan käyttää myös yhdessä Ajaxin kanssa.

4.1.4 Sovellustekniikan valinta

Air-sovellus voidaan toteuttaa käyttämällä edellä esitettyjä tekniikoita itsenäisesti. Yhdistelemällä tekniikoita sovellukseen voidaan valita tekniikoiden parhaat ominaisuudet. Valittavaan sovellustekniikkaan vaikuttavia tekijöitä pitää miettiä aina tapauskohtaisesti. Flash sopii erityisesti paljon multimediaa sisältäviin sovelluksiin, kuten peleihin ja opetusohjelmiin. Valinta HTML/Ajax- ja Flex -tekniikoiden välillä on monimutkaisempaa.

HTML/Ajax-tekniikan valintaan vaikuttaa varmasti kyseisten tekniikoiden tuntemus ja osaaminen. Nykyinen web-sovellus on kohtalaisen helppo siirtää Air-sovellukseksi vähäisin muutoksin. Monimutkaisten käyttöliittymän toteuttaminen HTML/Ajax-tekniikalla voi kuitenkin olla aikaa vievää. Nykyisin on kuitenkin tarjolla monia käyttöliittymäkomponentteja sisältäviä Ajax-kirjastoja, jotka ovat yhteensopivia Adobe Air:n kanssa. Esimerkisi ExtJs 2.0 on erittäin monipuolinen vaihtoehto käyttöliittymän toteutukseen (Kuva 4).



Kuva 4. Esimerkki Adobe Air- ja ExtJS 2.0 -sovelluksesta

Flex on vielä monelle tuntematon tekniikka. Syynä on edeltävien versioiden kallis hinta, joka vaikutti tekniikkaan tutustumiseen negatiivisesti. Helmikuussa 2008 julkaistu Flex 3 SDK on ilmainen, mikä lisää mielenkiintoa tekniikkaa kohden. Flex sisältää paljon valmiita käyttöliittymäkomponentteja, joiden hyödyntäminen nopeuttaa sovelluskehitystä. Flex-sovellus käännetään binääri swf-tiedostoksi, joten suorituskyky on ajonaikaisesti tulkattavia HTML/Ajax-sovelluksia parempi. Täysin uuden sovelluksen kehittämiseen Flex on varmasti varteenotettava vaihtoehto. Flex-sovellukset ohjelmoidaan ActionScriptillä, joka on aidosti oliopohjainen. Flex-sovelluksissa voidaan käyttää helposti näyttäviä visuaalisia tehosteita ja multimedian lisääminen on yhtä helppoa kuin Flash-tekniikkaa käytettäessä (Kuva 5). [21.]

Kuva 5. CommuniGate Pronto! Flex-sovellus

4.1.5 *Air-sovelluksen kuvaustiedosto*

Air-sovelluksen tärkein tiedosto on XML-kuvaustiedosto, joka on pakollinen jokaisessa Air-sovelluksessa sovellustekniikasta riippumatta. Ilman kuvaustiedostoa Air-sovellusta ei voi paketoida asennettavaksi sovellukseksi, eikä edes käynnistää. Kuvaustiedoston avulla määritetään sovelluksen tärkeimmät ominaisuudet ja tiedot. Sovelluskehittimet luovat kuvaustiedoston automaattisesti uuden projektin yhteydessä. Komentorivityökaluja käytettäessä kuvaustiedosto pitää tehdä itse. Seuraavassa esitetään osissa kuvaustiedoston tärkeimmät ja pakolliset kohdat.

```
<?xml version="1.0" encoding="utf-8" ?>
<application xmlns="http://ns.adobe.com/air/application/1.0">
    ...
</application>
```

Kaikki Air-sovellusta kuvaavat elementit sijoitetaan `application`-juurielementin sisään. Juurielementille on määritetty Air-sovelluksen nimiavaruus `xmlns`-ominaisuudella. Nimiavaruuden viimeinen numerosarja ilmoittaa vaadittavan Air-ajoympäristön version, jonka sovellus vaati toimiakseen.

Sovelluksen tunnistetiedot

Ajoympäristö vaatii Air-sovellukselta tunnistetiedot asennusta ja versionhallintaa varten. Tunnistetietojen avulla voidaan tunnistaa sovelluksen nimi, julkaisija ja versio.

```
...
<id>fi.storebrowser</id>
<filename>StoreBrowser</filename>
<name>StoreBrowser</name>
<version>v1.0.0</version>
<copyright>Tuomas Haimi, 2007</copyright>
...
```

Tunniste määritetään `id`-elementillä. Se voi sisältää alfanumeerisia merkkejä ja suositeltavana merkintätapana käytetään käänteistä DNS-nimeämistä. Sovelluksen tiedostonimi määritetään `filename`-elementillä, ilman pistettä ja tiedostopäätettä. `Name`-elementillä määritetään sovelluksen nimi, joka näytetään sovelluksen asentamisen yhteydessä. Sovelluksen versio ilmaistaan `version`-elementillä. Versioinnin mielekkyys on sovelluskehittäjän vastuulla. Sovelluksen julkaisija voidaan ilmaista `copyright`-elementillä.

Asennustietojen määrittäminen

Sovelluskehittäjä voi määrittää sovelluksen ensisijaisen asennushakemiston kuvaustiedoston avulla. Käyttäjä voi kuitenkin vaihtaa hakemiston haluamakseen asennuksen yhteydessä.

```
...
<installFolder>StoreBrowser</installFolder>
<programMenuFolder>StoreBrowser</programMenuFolder>
...
```

Ensisijainen asennushakemisto määritetään `installFolder`-elementillä. Windows-käyttöjärjestelmässä ensisijainen hakemisto on *C:\Program Files\StoreBrowser*. Mac OS X -käyttöjärjestelmässä */Applications /StoreBrowser* -hakemistoon. Windows-käyttöjärjestelmää varten voidaan myös määrittää pikakuvakkeen sijainti ohjelmat-valikossa `programMenuFolder`-elementillä. Muilla käyttöjärjestelmillä ominaisuus ei ole käytössä.

Sovellusikkunan määrittäminen

Air-sovellukset ovat työpöytäsovelluksia. Sovellusikkunan sisältö, koko, sijainti näytöllä sekä muut ominaisuudet ovat tarkasti määriteltävissä kuvaustiedoston avulla.

```
...
<initialWindow>
    <content>root.html</content>
    <title>Asiakasselain</title>
    <systemChrome>standard</systemChrome>
    <transparent>false</transparent>
    <visible>true</visible>
    <minimizable>true</minimizable>
    <maximizable>true</maximizable>
    <resizable>true</resizable>
    <width>1280</width>
    <height>786</height>
    <x>0</x>
    <y>0</y>
</initialWindow>
...
```

Ominaisuudet määritetään `initialWindow`-elementin lapsisolmuina. Sovelluksen varsinainen sisältö määritetään `content`-elementillä, joka on sovellustekniikasta riippuen joko HTML- tai SWF-tiedosto. Ikkunan otsikko määritetään `title`-elementillä. Sovellusikkunan kehysten näkyvyys määritetään `systemChrome`-elementillä. Oletusarvona on `standard`, jolloin käyttöjärjestelmän tarjoamat kehykset ovat näkyvissä. Asettamalla elementti `none`-arvoon, sovellusikkunan muoto ja koko on vapaasti muokattavissa. Sovelluksen läpinäkyvyyttä voidaan kontrolloida `transparent`-elementillä. Ikkuna voi olla läpinäkyvä ainoastaan silloin kun ikkunan kehykset eivät ole käytössä. Läpinäkyvyyden käyttö voi vaatia enemmän resursseja tietokoneelta kuin normaalia ikkunaa käytettäessä. Ikkunan näkyminen käynnistyksen yhteydessä voidaan määrittää `visible`-elementillä. Ikkuna on välittömästi näkyvissä `true`-arvolla. Vaihtoehtoisesti voidaan valita arvoksi `false`, jolloin ikkuna ei ole näkyvissä. Sovellusikkunan näkyvyyttä voidaan kontrolloida myös sovelluksesta käsin. Sovellusikkunan koko, sijainti sekä minimi- ja maksimikoko voidaan määrittää `width`-, `height`-, `x`-, `y`-, `minsize`- ja `maxsize`-elementeillä. Arvoiksi annetaan kyseistä kokoa tai sijaintia vastaavana pikselien lukumääränä. Sovellusikkunan pienentäminen ja suurentaminen voidaan estää tai sallia `minimizable`, `maximizable`- ja `resizable`-elementeillä. Elementtien oletusarvoina on `true`.

Tiedostotyyppien määrittäminen

Air-sovellukselle voidaan määrittää ne tiedostotyypit, jotka sovellus tunnistaa automaattisesti. Sovellus voidaan määrittää myös tietyn tyyppisten tiedostojen ensisijaiseksi sovellukseksi, jolla tiedostoja käsitellään. Käyttöjärjestelmä rekisteröi tiedostotyypit asennuksen yhteydessä Air-sovellukselle, jos niitä ei ole vielä rekisteröity toiselle sovellukselle.

```
...
<fileTypes>
    <fileType>
        <name>Air.Image</name>
        <extension>jpg</extension>
        <description>Air image file</description>
        <contentType>img/jpeg</contentType>
        <icon>
            <image16x16>icons/A_16.png</image16x16>
            <image32x32>icons/A_32.png</image32x32>
            <image48x48>icons/A_48.png</image48x48>
            <image128x128>icons/_128.png</image128x128>
        </icon>
    </fileType>
</fileTypes>
...
```

Tiedostotyyppien määrittäminen ei ole pakollista. Tiedostotyypit määritetään `fileTypes`-elementin `fileType`-lapsisolmuiksi. Jokaiselle tiedostotyypille pitää määrittää vähintään nimi `name`-elementillä ja tiedostopääte `extension`-elementillä. Tiedostotyypin kuvaus, sisältötyyppi ja ikonien määrittäminen on vapaaehtoista, mutta niiden määrittäminen auttaa käyttöjärjestelmää päättelemään paremmin tiedostotyypin ja sen käsittelyyn tarkoitetun sovelluksen.

Muut ominaisuudet

Sovellukselle voidaan valita työpöydällä ja valikoissa näkyvät ikonit. Jos ikoneita ei määritetä, käyttöjärjestelmä korvaa ikonit ajoympäristön vakioikoneilla. Tiedostomuotona hyväksytään ainoastaan PNG-kuvatiedostotyyppi.

```
...
<icon>
    <image16x16>icons/smallIcon.png</image16x16>
    <image32x32>icons/mediumIcon.png</image32x32>
    <image48x48>icons/bigIcon.png</image48x48>
    <image128x128>icons/biggestIcon.png</image128x128>
</icon>
...
```

Air-sovellus voidaan käynnistää WWW-sivulla sijaitsevalla linkillä. Käynnistäminen määritetään `allowBrowserInvocation`-elementillä, joka oletusarvo on `false`.

```
...
<allowBrowserInvocation>true</allowBrowserInvocation>
...
```

Air-ajoympäristön tarjoaa yhdenmukaisen ja käyttöjärjestelmästä riippumattoman asennus- ja päivitysprosessin. Vaihtoehtoisesti sovelluskehittäjä voi toteuttaa sovelluskohtaisen käyttöliittymän sovelluksen päivittämiseen ja huolehtia päivityksen toteuttamisesta haluamallaan tavalla.

```
...
<customUpdateUI>true</customUpdateUI>
...
```

Asettamalla `customUpdateUI`-elementin arvoksi `true`, ajoympäristölle ilmoitetaan sovelluksen huolehtivan päivityksestä. Päivittäminen edellyttää, että käyttäjän koneelle on asennettu kyseinen sovellus. Lisäksi päivitettävän sovelluksen ja päivityksen tunnisteiden pitää vastata toisiaan.

4.2 Tietoturva

Air-sovelluksen tietoturva poikkeaa jonkin verran Internet-selaimen tietoturvatoteutuksesta. Air-sovellus suoritetaan samoilla oikeuksilla normaalien työpöytäsovelluksien tapaan. Air-sovellus voi käyttää tietokoneen resursseja, kuten käsittelemään tiedostoja ja muodostamaan verkkoyhteyksiä. Air-sovellusta koskevat kuitenkin käyttöjärjestelmän käyttäjäkohtaiset oikeudet ja rajoitukset. Seuraavaksi esitetään tietoturvan kannalta oleellisia piirteitä. [22, s. 1.]

4.2.1 Tietoturvamalli

Air perustuu tietoturvamalliin, jossa jokaiselle tiedostolle määritetään oikeudet. Oikeudet määritetään ajonaikaisesti, perustuen tiedoston alkuperäiseen lähteeseen. Tiedostot jaetaan turvallisuusryhmiin, joita kutsutaan hiekkalaatikoiksi. Hiekkalaatikot ja niihin kuuluvat tiedostot ovat oletusarvoisesti erotettu toisistaan. Tietoturvamalli on kehitetty Flash Playerin tietoturvamallista, lisättynä `application`-hiekkalaatikolla. Hiekkalaatikoiden avulla erotellaan luotettavat ja epäluotettavat tiedostot toisistaan. Epäluotettava tiedosto ei voi käsitellä Air-rajapintoja ja tietokoneen resursseja suoraan. Hiekkalaatikoihin perustuva tietoturvamalli mahdollistaa eri lähteistä ladatun sisällön turvalli-

sen käytön Air-sovelluksessa. Seuraavassa on esitetty lyhyesti eri hiekkalaatikot ja niiden merkitys. [22, s. 5.]

Application

Tiedostoilla, jotka kuuluvat `application`-hiekkalaatikkoon, on täydet oikeudet käyttää ajoympäristön tarjoamia palveluita ja rajapintoja. Hyödyllisin näistä on tietokoneen paikallisten tiedostojen ja hakemistojen käsittely. Air-sovellushakemistossa sijaitsevat tiedostot kuuluvat automaattisesti `application`-hiekkalaatikkoon. [22, s.5]

Non-application

Tiedostot, jotka ladataan sovellukseen ulkopuolelta kuuluvat `non-application` -hiekkalaatikoihin, jotka on jaettu neljään ryhmään sen perusteella, mistä tiedosto on ladattu ja minkä tasoiset oikeudet tiedostoilla on. Tiedosto, joka ladataan Air-sovellukseen Internet-sivulta, kuuluu `remote` -hiekkalaatikkoon. Eri lähteistä, eli Internet-domaineista ladatut tiedostot, sijoitetaan omiin hiekkalaatikoihin. Eri hiekkalaatikoissa olevat tiedostot eivät voi vaikuttaa toisiinsa ja ne eivät voi suoraan hyödyntää ajoympäristön rajapintoja. Samasta lähteestä olevat tiedostot puolestaan voivat käsitellä ja kutsua toistensa palveluita. Käyttäjän turvalliseksi hyväksymä paikallinen tiedosto kuuluu `local-trusted` -hiekkalaatikkoon. Tiedoston oikeuksia on rajoitettu, mutta se voi kuitenkin käsitellä paikallisen tietokoneen tiedostoja ja muodostaa verkkoyhteyksiä. `Local-with-networking` -hiekkalaatikkoon kuuluvilla tiedostoilla on oikeus käyttää verkkoyhteyksiä. Käyttäjä ei ole kuitenkaan merkinnyt tiedostoja luotettaviksi, joten tiedostot eivät voi käsitellä paikallisia resursseja. Tähän hiekkalaatikkoon voivat kuulua ainoastaan SWF-tiedostot. `Local-with-filesystem` -hiekkalaatikkoon kuuluvat tiedostot voivat käsitellä paikallisia resursseja ja tiedostoja, mutta eivät voi käyttää verkkoyhteyksiä. Käyttäjä ei ole merkinnyt tiedostoja luotettaviksi. [24, s. 13-15.]

4.2.2 Sovelluksen allekirjoittaminen

Tietoturvan parantamiseksi kaikki Air-sovellukset pitää allekirjoittaa digitaalisesti. Allekirjoituksella varmistetaan sovelluksen valmistaja ja sovelluksen eheys. Sovellus voidaan allekirjoittaa luotettavan tahon myöntämällä tai itse luodulla sertifikaatilla. Adobe Air-ajoympäristö tunnistaa Verisign ja Thawten myöntämät sertifikaatit, jolloin sovellusta asennettaessa sovelluksen julkaisija voidaan varmistaa. Käytettäessä itse luotua sertifikaattia Air-ajoympäristö näyttää asennuksen yhteydessä varoituksen, että sovelluksen julkaisijaa ei voida yksikäsitteisesti varmistaa. Allekirjoituksen avulla varmistetaan myös se, ettei sovellusta ole muutettu allekirjoituksen jälkeen. Sertifikaatin luonti ja sovelluksen allekirjoittaminen SDK:n mukana tulevilla työkaluilla käsitellään luvussa 5.2.1. [22, s. 4.]

4.2.3 Asennus ja päivitys

Air-sovellus levitetään `.air`-asennustiedostoina. Sovellukset voidaan asentaa suoraan Internet-sivuilta, asennuslevyltä tai esimerkiksi muistikortilta. Ajoympäristö vastaa sovelluksen asentamisesta käyttäjän tietokoneelle. Sovelluskehittäjä voi määrittää sovellukselle nimen ja version, mutta ei voi ohjelmallisesti vaikuttaa itse asennusprosessin eri vaiheisiin. Air-ajoympäristö tarjoaa käyttöjärjestelmäriippumattoman asennusprosessin ja näkymän. Käyttäjä sallii sovelluksen asentamisen ja valitsee sovelluksen asennuskohteen. Asennuksen yhteydessä näytetään tietoja sovelluksen tekijästä ja sovelluksen lähde, jos asennustiedosto on allekirjoitettu luotettavaksi hyväksytyn tahon myöntämällä sertifikaatilla. [22, s. 2 –3.]

Air-sovelluksen asentamisen yhteydessä ajoympäristö tarkistaa, onko sovellus jo asennettu käyttäjän tiekoneelle. Ajoympäristö vertaa sovelluksen versionumeroa. Koska ajoympäristö vertaa ainoastaan versionumeroa, voi suuremman version omaava sovellus olla vanhempi kuin tietokoneelle asennettu versio. Versiointi ja sen mielekkyys on sovelluskehittäjän vastuulla. Ajoympäristö sisältää mahdollisuuden tarkistaa sovelluksen käynnistämisen yhteydessä, onko saatavilla uudempaa versiota sovelluksesta ja tarvittaessa päivittää sovelluksen. [22, s. 2 – 3.]

4.3 Adobe Airin edut ja heikkoudet

Adobe Air tarjoaa paljon uusia ja mielenkiintoisia mahdollisuuksia toteuttaa entistä näyttävämpiä työpöytäsovelluksia, jotka osittain myös tukeutuvat sovellus- ja tietokantapalvelimien tarjoamiin palveluihin. Rikkaat Internet-sovellukset ovat hämärtäneet web- ja työpöytäsovellusten välistä eroa. Rikkaat Intenet-sovellukset muistuttavat toiminnaltaan ja ulkoasultaan erehdyttävästi normaaleja työpöytäsovelluksia ja Air-tekniikan myötä työpöydälle voidaan toteuttaa rikkaita sovelluksia perinteisillä web-tekniikoilla. Seuraavassa esitetään Air-sovellusten etuja ja heikkouksia sekä vertaillaan rikkaiden Internet-sovellusten eroja Internet-selaimessa ja työpöydällä.

Edut

Air-sovellusten etuna on käyttöjärjestelmäriippumattomuus. Air-sovellus toteutetaan suoraan ajoympäristölle, joten eri käyttöjärjestelmät eivät vaadi omaa versiota sovelluksesta. Sovellus toimii kaikilla käyttöjärjestelmillä, joille ajoympäristö on saatavana. Ajoympäristön tehtävänä on huolehtia sovelluksen yhtenäisestä toiminnasta ja ulkoasusta, sekä piilottaa käyttöjärjestelmien erot sovelluskehittäjältä. Perinteiseen web-sovelluskehitykseen verrattuna Air-sovellusta ei tarvitse suunnitella ja testata useilla eri valmistajien selaimilla. Se säästää huomattavasti aikaa ja yksinkertaistaa sovelluskehitystä.

Perinteisten web-sovelluksien parissa työskennellyt henkilö voi suoraan aloittaa Air-sovelluskehityksen ja hyödyntää täysin osaamistaan. Air mahdollistaa monipuolisten ja näyttävien työpöytäsovelluksien toteuttamisen erittäin helposti. Yhdistelemällä HTML-, Flash- ja Flex -tekniikoiden parhaita puolia voidaan luoda dynaamisia ja interaktiivisia työpöytäsovelluksia. Sovellusten käytettävyys on kehittyneempää kuin perinteisissä web-sovelluksissa. Tekniikoita yhdistelemällä Air-sovellukseen voidaan helposti liittää multimediaa, kuten ääntä, kuvia, videoita, vektorigrafiikkaa ja 3D-animaatiota.

Heikkoudet

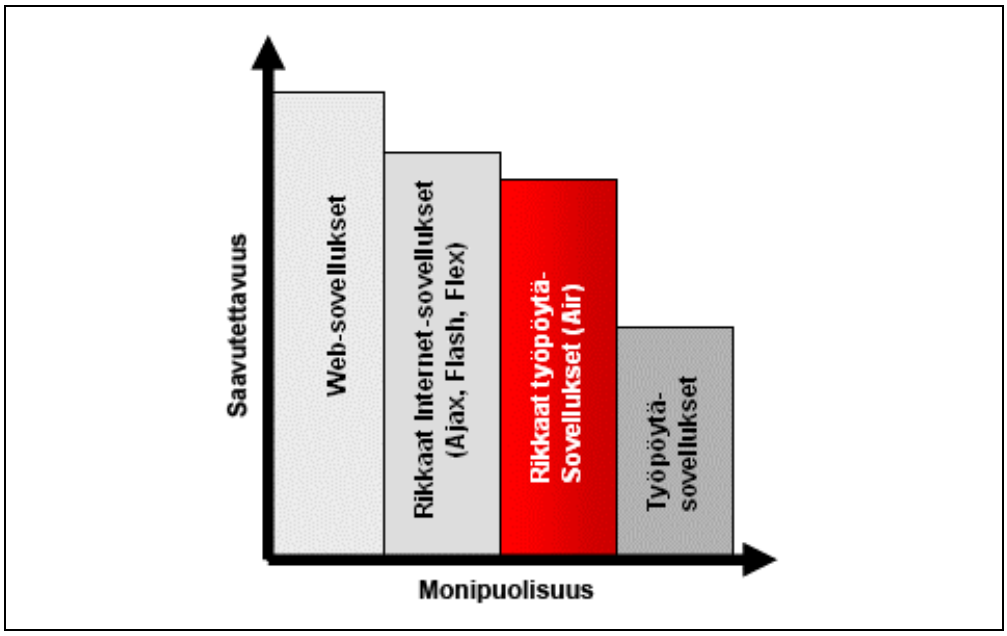
Air-sovellusten ilmiselvänä heikkoutena on tällä hetkellä ajoympäristön levinneisyyden rajoittuneisuus. Linux ja mobiililaitteiden tuen puuttuminen rajoittavat Air-tekniikan leviämistä. Air-sovellukset eivät pysty tarjoamaan samaa saavutettavuutta kuin selaimessa toimivat rikkaat Internet-sovellukset, joiden käyttöön riittää Internet-selaimella varustettu päätelaite ja verkkoyhteys.

4.4 Internet-selain vai työpöytä

Internet-selain on noussut varteenotettavaksi ohjelmistoalustaksi perinteisten työpöytäsovellusten rinnalle. Internet-selaimissa toimivien sovellusten etuna on ennen kaikkea käyttöjärjestelmäriippumattomuus, aikaan ja paikkaan sitoutumattomuus sekä sovellusten tehokas levitys ja ylläpito. Lisäksi tekniikan kehittyminen on mahdollistanut uusien, työpöytäsovelluksia muistuttavien sovellusten toteuttamisen. Adobe Air -ajoympäristö tuo rikkaat Internet-sovellukset työpöydälle yhdistäen web-sovellusten ja työpöytäsovellusten ominaisuuksia. Seuraavassa vertaillaan lyhyesti Internet-selaimessa toimivan sovelluksen ja Air-työpöytäsovelluksen ominaisuuksia. [25.]

Sovelluksen asennus ja päivitys

Internet-selaimessa toimiva sovellus on helppo levittää nopeasti useille käyttäjille ilman erillistä asennusprosessia. Päivitykset suoritetaan ainoastaan web-palvelimelle ja uusi versio sovelluksesta on kaikkien käyttäjien saatavilla. Air-työpöytäsovellus voidaan asentaa suoraan selaimesta tai kuten normaalit työpöytäsovellukset. Sovelluksen päivittämiseen Air sisältää rajapinnan, jonka avulla sovellus voi päivittää itsensä uuden päivityksen ollessa tarjolla. Pääsääntöisesti työpöytäsovellukset ovat Internet-selaimessa suoritettavia sovelluksia tehokkaampia ja monipuolisempia. Web-sovelluksen saavutettavuus on kuitenkin huomattavasti parempi kuin työpöytäsovelluksen (Kuva 6). [25; 26.]



Kuva 6. Saavutettavuuden ja monipuolisuuden suhde [lähdettä 26 mukaillen]

Käyttöjärjestelmätuki

Internet-selain löytyy käytännössä lähes kaikista käytössä olevista käyttöjärjestelmistä. Web-sovellukset ovat käyttöjärjestelmäriippumattomia. Osa rikkaiden Internet-sovellusten tekniikoista vaatii kuitenkin erillisen liitännäisen, jota ei välttämättä ole tarjolla jokaiselle käyttöjärjestelmälle. Air-ajoympäristö on tällä hetkellä saatavilla ainoastaan Windows- ja Mac OS X-käyttöjärjestelmille. Tuki Linux-käyttöjärjestelmille on tulossa myöhemmin. [25.]

Työpöytäintegraatio

Internet-selaimessa toimivat sovellukset suoritetaan rajoitetussa hiekkalaatikossa, eivätkä ne voi käsitellä tietokoneen paikallisia resursseja. Internet-selain sisältää oman käyttöliittymän, joka sisältää nykyisten sovellusten kannalta turhia ja häiritseviä toimintoja. Käyttöliittymää ei voi muokata mieleisekseen. Lisäksi sovelluksen toiminta rajoittuu aktiiviseen selainikkunaan. Air-sovellus suoritetaan samoilla oikeuksilla kuin muutkin työpöytäsovellukset. Tietokoneen resurssien käyttöä ei ole rajoitettu. Air-sovelluksen käyttöliittymää voidaan muokata vapaasti ja myös taustalla toimivat, ilman näkyvää käyttöliittymää olevat sovellukset, ovat mahdollisia. [25.]

Tiedonvarastointi

Internet-selaimet sisältävät hyvin rajoitetun mahdollisuuden tallentaa sovellustiedostoja käyttäjän tietokoneelle. Air-sovellus puolestaan voi tallentaa, luoda, muokata ja poistaa tiedostoja. Lisäksi ajoympäristö sisältää sisäänrakennetun tietokannan, jota sovellukset voivat hyödyntää. [25.]

Verkkotoiminnot ja offline-tuki

Internet-selaimessa suoritettava sovelluksen käyttö keskeytyy verkkoyhteyden katkettua palvelimeen ja yleensä senhetkiset tiedot menetetään. Tarjolla on kuitenkin liitännäisiä, joiden avulla offline-tuki voidaan lisätä sovellukseen. Internet-selaimen tietoturvamalli estää selainta luomasta uusia yhteyksiä muihin verkkoalueisiin. Yhteydet ovat sallittu ainoastaan samaan verkkoalueeseen, josta sovellus alunperin ladattiin. Työpöytäsovellukset käyttävät pääsääntöisesti paikallisia resursseja, joten verkkoyhteyden katkeaminen ei keskeytä sovellusten toimintaa. Air-sovelluksessa verkkoyhteyksien luontia ei ole estetty. Air-ajoympäristö sisältää myös toiminnon, jonka avulla sovellus voi tarkkailla verkkoyhteyksiä ja toimia sen edellyttämällä tavalla. [25.]

4.5 Kilpailevat tekniikat

Adobe Air -sovellukset ovat työpöytäsovelluksia, jotka kuitenkin toteutetaan perinteisillä web-tekniikoilla. Air yhdistää työpöytä- ja web-sovelluksen ominaisuuksia. Uudet Internet-tekniikat tekevät kuitenkin rikkaista Internet-sovelluksista varteenotettavia vaihtoehtoja työpöytäsovelluksille. Kilpailu eri tekniikoiden välillä on kiristynyt ja uusia tekniikoita julkaistaan kasvavalla vauhdilla. Seuraavassa esitellään lyhyesti muutamia tämänhetken varteenotettavia tekniikoita. Osittain tekniikat ovat tyypiltään erilaisia ja kilpailevat eri markkinaosuuksista, selkeä ero web- ja työpöytäsovellustekniikoiden välillä alkaa hämärtyä.

Ajax

Ajax voidaan käsittää uudenlaisena tapana toteuttaa web-sovelluksia. Ajax on myös yhdistelmä tekniikoita, joiden avulla voidaan luoda perinteisiä web-sovelluksia dynaamisempia ja interaktiivisempia sovelluksia. Ajax mahdollistaa WWW-dokumentin osittaisen päivittämisen ilman koko sivun uudelleen lataamista. Tämä lisää web-sovelluksen interaktiivisuutta, nopeutta ja käytettävyyttä. Web-sovellukset alkavat muistuttaa toiminnaltaan ja ominaisuuksiltaan yhä enemmän työpöytäsovelluksia. [9.]

Microsoft kehitti XMLHttpRequest-olion kaltaisen konseptin osaksi Outlook Web Access 2000:tta varten. Microsoftin XMLHTTP toteutus on ollut käytössä Internet Explorer 5:stä asti. Vuonna 2002 Mozilla Projekti toteutti oman XMLHttpRequest-olion. Tuki levisi nopeasti myös Apple Safarin ja Operan selaimiin. [27.]

W3C julkaisi toukokuussa 2006 ehdotuksen XMLHttpRequest-spesifikaatiosta. Sen tarkoitus on yhtenäistää selainten yhteensopivuutta XMLHttpRequest-olion käsittelyssä. Tällä hetkellä Internet Explorer 7 tukee XMLHTTP- ja XMLHttpRequest-oliota, joten yhteensopivuusongelmasta on päästy osittain eroon Internet-selainten uusiutumisen myötä. [28.]

Mozilla Prism

Prism on Mozilla Foundationin julkaisema tekniikka, joka siirtää web-sovellukset suoritettavaksi omassa ikkunassaan ilman selainohjelmistoa. Sen tarkoitus on yksinkertaistaa sovelluksen käyttöliittymää poistamalla nykyisten web-sovelluksen kannalta turhat navigointipainikkeet ja osoitekentän. Omassa erillisessä ikkunassa suoritettava sovellus muistuttaa enemmän perinteistä työpöytäsovellusta. Käytännössä sovellus kuitenkin suoritetaan kuitenkin selainohjelmistossa. Prism ei sisällä yhtä syvää työpöytäintegraatiota kuin esimerkiksi Adobe Air. Sovellukselle voidaan luoda pikakuvake työpöydälle ja käynnistää sovellus suoraan pikakuvakkeesta. Hyödyllisin ominaisuus Prism-tekniikassa on web-sovellusten suorittaminen omissa prosesseissa, jolloin yhden sovelluksen jumiutuminen ei kaada muita toiminnassa olevia sovelluksia. [29.]

Microsoft Silverlight

Microsoft Silverlight on .NET-tekniikkaan perustuva liitännäinen, joka mahdollistaa entistä näyttävämpien rikkaiden Internet-sovellusten ja esimerkiksi pelien toteuttamisen Internet-selaimille. Silverlight kilpailee etenkin Adoben Flash- ja Ajax-tekniikoiden kanssa, mutta vaatii selaimeen asennettavan liitännäisen. Silverlight tarjoaa Flashia paremman suorituskyvyn ja monipuolisemmat kirjastot mediankäsittelyyn ja käyttöliittymien toteuttamiseen. Käyttöliittymät kuvataan XML-pohjaisella XAML-merkintäkielellä. Samalla tekniikalla toteutetaan myös .NET Windows -sovellukset. [30.]

Silverlight-sovellusten kehittämiseen voidaan käyttää Microsoftin tulevaa Visual Studio ”Orcas” -kehitystyökalua. Kolmannen osapuolen kehitystyökaluja ei ainakaan vielä ole saatavilla. Silverlightin heikkoutena on ainakin vielä liitännäisen huono levinneisyys ja toiminta ainoastaan Windows- ja Mac OS X -käyttöjärjestelmissä. [30.]

OpenLaszlo

OpenLaszlo on ilmainen avoimen lähdekoodin ohjelmointialusta rikkaiden Internet-sovellusten toteuttamiseen. Sovellukset toteutetaan XML-kieleen pohjautuvalla LXZ-merkintäkielellä ja JavaScriptillä. OpenLaszlo-sovellusten toteuttaminen muistuttaa hyvin paljon web-sovellusten toteuttamista HTML- ja JavaScript-kielillä. LXZ-merkintäkielellä ohjelmoitu sovellus voidaan kääntää Ajax-sovellukseksi tai Flash Player -sovelluksella suoritettavaksi SWF-tiedostoksi. [16.]

Sun JavaFX

JavaFX on Sun Microsystemin toukokuussa 2007 esittelemä uusi rikkaiden Internet-sovellusten kehittämiseen tarkoitettu teknologia ja tuoteperhe. JavaFx:n ideana on toteuttaa sovellukset siten, että yksi ja sama sovellus toimii kaikissa päätelaitteissa. JavaFX koostuu kahdesta osasta: JavaFX Scriptistä ja JavaFX Mobilesta, jotka ovat avoimen lähdekoodin tuotteita. JavaFX Script on skriptikieli sovellusten toteuttamiseen ja JavaFX Moblile on mobiili- ja pda-laitteiden käyttöjärjestelmä. JavaFx-sovellusten suurimpana ongelmana on niiden vaatiman Java-virtuaalikoneen raskas toteutus. On todettu, että JavaFX on hitaampi kuin esimerkiksi Microsoftin Silverlight. [30; 31.]

5 ADOBE AIR: ASENNUS, KÄYTTÖÖNOTTO JA KEHITYSTYÖKALUT

Adobe Air on tällä hetkellä saatavana Windows XP-, Vista- ja Mac OS X-käyttöjärjestelmille. Air-sovelluksen käyttö edellyttää Adobe Air -ajoympäristön asentamista. Sovelluskehitystä varten tarvitaan lisäksi Adobe Air SDK, joka sisältää tarvittavat komentorivityökalut ja esimerkkisovelluksia. Laajempia sovelluksia toteutettaessa suositellaan asentamaan Air-tuen sisältävä sovelluskehitin. Seuraavassa esitetyt ohjeet on tarkoitettu Windows XP -käyttöjärjestelmälle.

5.1 Asennus

5.1.1 Ajoympäristön asennus

Ajoympäristön asennus aloitetaan lataamalla asennustiedosto osoitteesta http://get.adobe.com/air/. Tällä hetkellä uusin versio on helmikuussa 2008 julkaistu Adobe Air 1.0. Ajoympäristön koko on noin 11 megatavua, ja se on huomattavan pienikokoinen verrattuna muihin ajoympäristöihin. Latauksen jälkeen käynnistetään asennusohjelma. Asennus on helppo, eikä vaadi käyttäjältä toimenpiteitä. Ajoympäristö on käyttövalmis asennuksen jälkeen ja toimii taustalla ilman erillistä käyttöliittymää ja käyttäjän toimia. Asennuksen toimivuuden voi tarkistaa lataamalla samasta osoitteesta jonkin esimerkkisovelluksen.

Järjestelmävaatimukset

Suositellut järjestelmävaatimukset Windows-käyttöjärjestelmällä varustetuille tietokoneille on Intel Pentium 1GHz:n prosessori ja 512 megatavua muistia. Tällä hetkellä ajoympäristö on saatavilla Microsoft Windows 2000 SP4:n, Windows XP SP2:n ja Windows Vistan eri versioille sekä Mac OS X v10.4.910 versiota uudemmille. Mac OS X -käyttöjärjestelmän vaatimuksena on vähintään PowerPC G4 1GHz- tai Intel Core Duo 1,83 GHz:n prosessori ja 512 megatavua muistia. Sovellus, joka käyttää koko ruudun peittävää videokuvaa, voi vaatia tietokoneelta enemmän suorituskykyä. Tällä hetkellä ajoympäristö tukee ainoastaan englannin kieltä. [32.]

5.1.2 SDK:n asennus ja käyttöönotto

Sovelluskehitystä varten tietokoneelle pitää asentaa Adobe Air SDK. Se sisältää tarvittavat komentorivityökalut ja esimerkkiohjelmia helpottamaan sovelluskehitystä. Asennus aloitetaan lataamalla SDK osoitteesta http://www.adobe.com/products/air/tools/sdk/. Uusin versio on Adobe Air SDK 1.0.

Latauksen jälkeen puretaan asennuspaketti haluttuun hakemistoon. Käytetään hakemistosta nimeä <*SDK_hakemisto*>. Air SDK sisältää kaksi komentorivityökalua, joita käytetään Air-sovelluksen testaamisen ja paketoimiseen. Käytön helpottamiseksi kannattaa asettaa <*SDK_hakemisto*>*/bin*-hakemisto järjestelmäpolkuun. Windowsin ohjauspaneelista valitaan järjestelmä ja lisäasetukset-välilehti. Ympäristömuuttujat-nappia painamalla avautuu uusi ikkuna, josta valitaan PATH ja painetaan muokkaa-nappia. Rivin loppuun lisätään viittaus <*SDK_hakemisto*>*/bin* -hakemistoon.

Järjestelmäpolun muokkauksen jälkeen asennuksen toimivuutta voidaan testata käynnistämällä komentorivin ja antamalla komennon *adt*. Asennus on onnistunut, jos Windows tunnistaa ja suorittaa komennon.

5.2 Kehitystyökalut

Tehokkaat työkalut ovat tärkeässä asemassa sovelluskehityksessä ja testauksessa. Erityisesti graafiset sovelluskehittimet ovat nopeuttaneet ja helpottaneet sovelluskehitystä entisestään. Ne sisältävät monia ohjelmointityötä tehostavia ominaisuuksia. Automaattiset koodin täydennys-, virheidentunnistus- ja koodinvärjäysominaisuudet alkavat olla jokaisen kehitystyökalun perusominaisuuksia. Komponenttipohjaisten käyttöliittymien suunnittelun avuksi on tullut tehokkaita suunnittelutyökaluja, joilla komponentit sijoitetaan suunnittelunäytössä paikoilleen ja sovelluskehitin luo käyttövalmiin ohjelmakoodin suunnitelman pohjalta. Adobe Airin yhtenä etuna ovat helppokäyttöiset ja tehokkaat työkalut sovellusten toteuttamiseen. Seuraavassa esitellään osa tarjolla olevista kehitystyökaluista.

5.2.1 Komentorivityökalut

Yksinkertaisen sovelluksen ohjelmointiin riittää hyvin pelkkä tekstieditori ja Air SDK:n sisältämät komentorivityökalut, joilla sovellus paketoidaan Air-sovellukseksi. Komentorivityökalut sopivat lähinnä sovellusten testaukseen. Laajempien sovellusten toteuttaminen komentorivityökaluilla voi olla monimutkaista ja aikaa vievää. Työkalujen toiminnan tunteminen auttaa kuitenkin ymmärtämään tarkemmin Air-sovelluskehityksen eri vaiheita.

Air Debug Launcher (ADL)

Sovellusta voidaan testata ADL-komentorivityökalulla. ADL mahdollistaa Air-sovelluksen käynnistämisen ilman asennusta tietokoneelle ja on siksi sopiva työkalu sovelluksen testaukseen. Sovellus käynnistetään hakemistossa, johon sovelluksen XML-kuvaustiedosto ja HTML- tai SWF-tiedosto on tallennettu. Käynnistyskomento on:

```
adl Helloworld.xml
```

Virhetilanteissa kannattaa ensimmäisenä varmistaa, että HTML/SWF- ja XML-tiedostot ovat samassa hakemistossa ja XML-kuvaustiedossa on viitattu oikeaan HTML/SWF-tiedostoon. Seuraavana kannattaa varmistaa, että XML-kuvaustiedostossa ei esiinny kirjoitusvirheitä ja XML-merkkkaus on hyvinmuodostettua.

Air Developer Tool (ADT)

Air-sovellus paketoidaan asennustiedostoksi, jos se halutaan asentaa käyttäjän tietokoneelle ja käynnistää työpöydältä normaalin työpöytäsovelluksen tapaan. Lisäksi paketointi pitää tehdä sovelluksille, joita levitetään esimerkiksi Internetin välityksellä. Air-asennustiedosto on zip-pohjainen tiedosto, joka sisältää kaikki tarvittavat tiedostot sovelluksen asentamiseen ja käyttöön. Yksinkertaisimmillaan se voi siis sisältää ainoastaan HTML/SWF-tiedoston ja XML-kuvaustiedoston.

Kaikki Air-asennustiedostot pitää allekirjoittaa. Allekirjoitus on osa Air-tietoturvakäytäntöä, jolla pyritään varmistamaan sovelluksen aitous. Testaus ja kehityskäyttöön voidaan käyttää itse tehtyjä sertifikaatteja. Yksinkertaisia sertifikaatteja voi tehdä ADT-komentorivityökalun avulla. Julkiseen levitykseen tarkoitetut sovellukset kannattaa allekirjoittaa tunnetun sertifikaatteja myöntävän yrityksen sertifikaatilla. Itse tehdyllä sertifikaatilla allekirjoitetut

sovellukset ilmoittavat sovelluksen julkaisijan tuntemattomaksi asentamisen yhteydessä. Sertifikaatti luodaan ADT-työkalulla seuraavasti:

```
adt -certificate -cn SelfSigned 1024-RSA signed.key
secret
```

Komento luo `signed.key` nimisen sertifikaattitiedoston. Kursivoidut parametrit määritetään itse. Salausalgoritmina käytetään 1024-, tai 2048-bittistä RSA-algoritmia. Varsinainen Air-asennustiedosto luodaan seuraavalla komennolla:

```
adt -package -certificate signed.key -password
secret HelloWorld.air
HelloWorld.xml HelloWorld.html
```

Komento luo `HelloWorld.air` -nimisen asennustiedoston, joka voidaan asentaa tietokoneelle. Asentamisen jälkeen sovellus voidaan käynnistää pikakuvakkeesta. Parametrina annetut sovelluksen tiedostot pitää määrittää niin, että XML-kuvaustiedosto annetaan aina ensimmäisenä.

5.2.2 Sovelluskehittimet

Graafiset sovelluskehittimet sisältävät paljon ohjelmointityötä tehostavia ja nopeuttavia ominaisuuksia. Kehitystyön alla oleva sovellus voidaan esimerkiksi käynnistää testausta varten yhdellä napin painalluksella. Laajemmat sovellukset saattavat sisältää satoja tiedostoja, joiden ylläpito ja muokkaus vaatii tarkoitukseen sopivat työkalut. Air-tuki on saatavilla Adoben suosituimpiin web-sovelluskehittimiin. Lisäksi on tarjolla myös erittäin monipuolinen ilmainen vaihtoehto sovelluskehittimeksi.

Adobe Dreamweaver

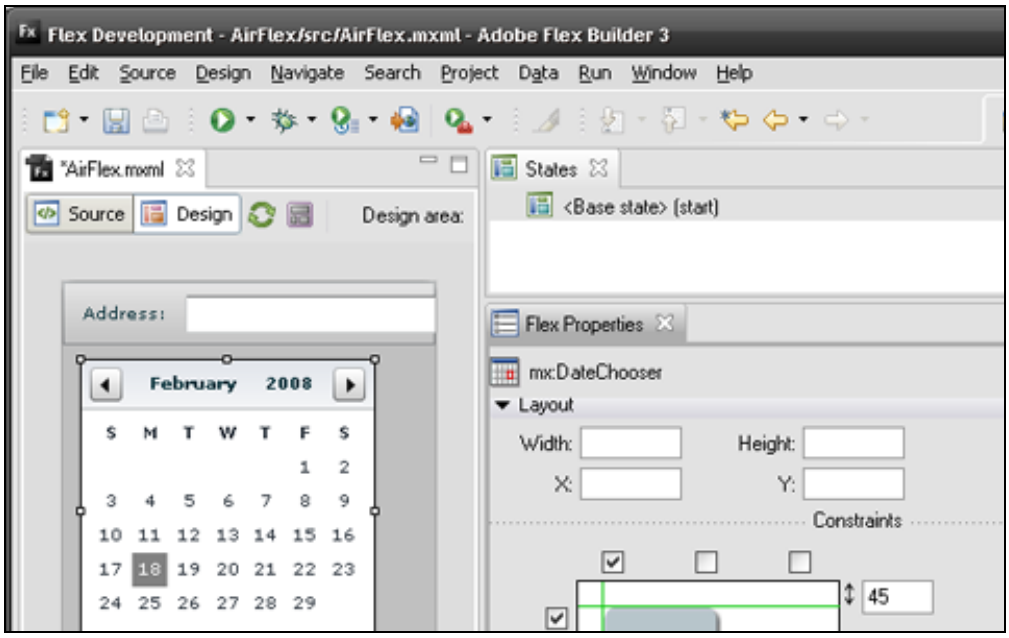
Adobe Dreamweaver CS3 on ammattikäyttöön suunniteltu maksullinen websivujen suunnittelu- ja muokkaustyökalu. Ohjelmasta löytyvät kaikki HTML-koodauksessa tarvittavat toiminnot. Dreamweaver sopii hyvin HTML- ja Ajax-sovellusten ohjelmointiin ja siihen on saatavilla lisäosa, joka mahdollistaa Air-sovellusten toteuttamisen. Lisäosan asentamisen jälkeen Dreamweaver osaa luoda Air-sovelluksessa tarvittavat tiedostot sekä tuottaa Air-asennustiedoston.

Adobe Flash

Adobe Flash CS3 on maksullinen, animaatiota ja vektorigrafiikkaa sekä multimediaa sisältävien web-sovellusten suunnittelu- ja kehitystyökalu. Air-sovellus voidaan toteuttaa kokonaan flash-tekniikalla, joka mahdollistaa graafisesti näyttävät multimediatyöpöytäsovellukset ja esimerkiksi yksinkertaiset pelisovellukset. Air-lisäosa on saatavana Adobe Flash CS3 -kehitystyökaluun, joten Air-sovellus voidaan toteuttaa suoraan ilman muita erillisiä työkaluja.

Adobe Flex Builder

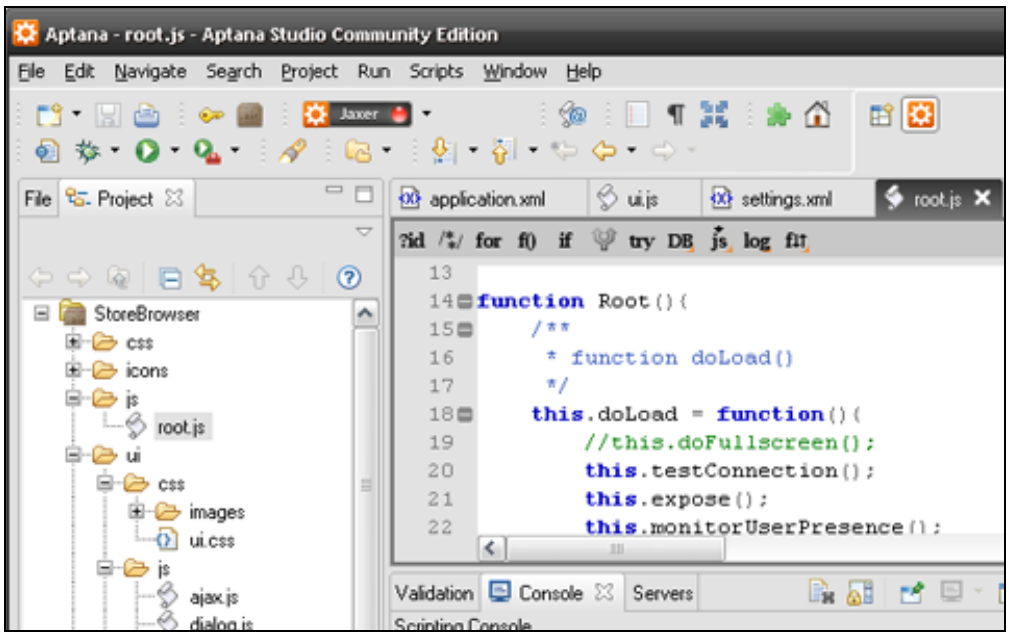
Adobe Flex Builder 3 on maksullinen Eclipse-pohjainen sovelluskehitin rikkaiden Internet-sovellusten toteuttamiseen Flex-tekniikalla (Kuva 7). Flex Builder sisältää erinomaisen käyttöliittymien suunnitteluominaisuuden, jonka avulla voidaan luoda nopeasti näyttäviä käyttöliittymiä monipuolisista käyttöliittymäkomponenteista. Flex Builder -sovelluskehittimeen on saatavana Air-lisäosa, jonka avulla voidaan helposti toteuttaa Air-sovellus Flex-tekniikalla. Flex 3 -sovelluskehys ja Flex Builder 3 julkaistiin samaan aikaan Adobe Air 1.0:n kanssa.



Kuva 7. Adobe Flex Builder 3

Aptana Studio

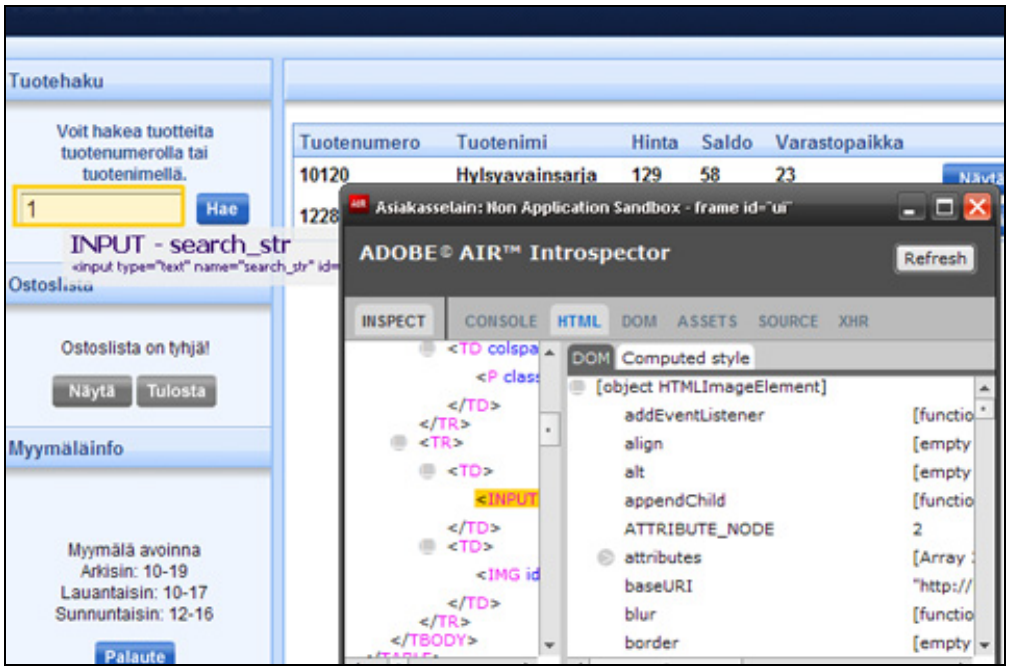
Aptana Studio 1.1 on ilmainen sovelluskehitin HTML- ja Ajax-sovellusten ohjelmointiin (Kuva 8). Aptana Studio tukee laajasti eri Ajax-kirjastoja, ja tuki Air-tekniikalle on saatavana lisäosana. Aptana sisältää ohjelmointityötä nopeuttavan koodin automaattisen täydentämisen ja värjäystoiminnon. Uusi Air-sovellus luodaan projektina ja Aptana luo kaikki Air-sovelluksessa tarvittavat tiedostot ja hakemistorakenteen automaattisesti.



Kuva 8. Aptana Studio 1.1

5.2.3 Testaustyökalut

Adobe Air SDK 1.0:n mukana tulee uusi ja paljon toivottu työkalu HTML- ja Ajax-sovellusten testaukseen ja virheiden jäljittämiseen. Air Introspector on lähes vastaava, kuin Firefox-selaimeen saatava Firebug (Kuva 9). Sen avulla voidaan tutkia sovelluksen rakennetta, DOM-solmupuuta, tyylitiedostoja, JavaScript-koodia, XMLHttpRequest-olion toimintaa ja palvelimen lähettämän vastauksen sisältöä ja otsakkeita. Lisäksi Air Introspector sisältää hyödyllisen konsoli-toiminnon, jonka avulla voi seurata tarkemmin sovelluksen toimintaa tulostamalla erilaisia varoitus- ja virheilmoituksia.



Kuva 9. Adobe Air Introspector-testaustyökalu

Air Introspector otetaan käyttöön lisäämällä sovellukseen määrittelyihin SDK:n mukana toimitettava Introspector-JavaScript -tiedosto. Air Introspector käynnistetään painamalla F12-näppäintä Air-sovelluksen ollessa käynnissä.

6 ADOBE AIR -SOVELLUS AJAX-TEKNIIKALLA

Insinöörityön tarkoituksena on esitellä Adobe Air -tekniikan tarjoamat mahdollisuudet työpöytäsovelluksen toteuttamisessa. Työtä varten suunniteltiin kohtalaisen laaja sovellus. Air-sovellus toteutettiin käyttämällä HTML- ja Ajax-tekniikoita. Tähän päädyttiin ensisijaisesti tekniikoiden tuntemuksen perusteella ja ilmaisen sovelluskehittimen takia. Perinteisiä web-tekniikoita käytettäessä ei jouduta käyttämään aikaa uuden ohjelmointikielen opiskeluun. Toisaalta Flex-tekniikan MXML-määrittelykieli muistuttaa hyvin paljon HTML-kieltä ja ActionScript puolestaan JavaScriptiä, joten uuden ohjelmointikielen opiskelu ei välttämättä vie kovin paljoa aikaa.

6.1 Sovelluksen esittely

Insinöörityön yhteydessä toteutettiin sovellus, jonka tarkoitus on korvata tavarataloissa tällä hetkellä laajasti käytössä oleva asiakaspääte. Sovelluksen kaikkia toimintoja ei ole ensimmäisessä versiossa toteutettu, sillä työn tarkoituksena oli tutkia tarkemmin Air-tekniikkaa ja sen mahdollisuuksia. Nykyinen käytössä oleva asiakaspäätesovellus on toteutettu käyttötarkoitusta varten muokatulla Internet-selaimella, jossa ajetaan web-sovellusta. Sovellus hyödyntää tavaratalon omaa varasto-tietokantaa, joka sisältää erilaisia tietoja tuotteista.

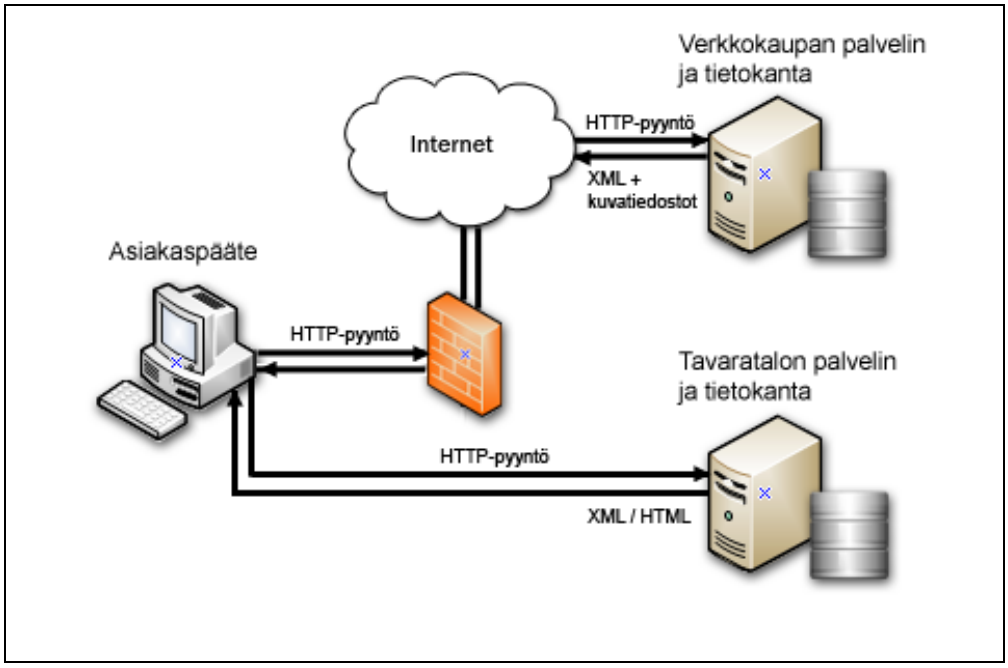
Nykyisen sovelluksen toimintoina on mahdollisuus etsiä tuotteita tuotenimen ja -numeron avulla. Tuotteesta näytetään hinta, varastotilanne sekä sijainti tavaratalossa. Tuotteita voidaan lisätä ostoslistaan ja tulostaa lista, jonka avulla tuotteiden löytäminen tavaratalosta helpottuu. Lisäksi asiakaspäätteellä voi jättää asiakaspalautetta tavaratalon henkilökunnalle. Asiakaspääte on höydyllinen lisäpalvelu tavaratalossa ja se toimii myös henkilökunnan apuna päivittäisessä työssä.

Asiakasselain

Air-tekniikalla toteutetun työpöytäsovelluksen on tarkoitus korvata asiakaspääte uutta tekniikkaa hyödyntäen. Asiakaspäätteen toimintoja ei ole tarkoitus lisätä alussa, mutta jatkossa lisätoimintoina voisi olla esimerkiksi sähköisen tuotekuvaston selaaminen asiakaspäätteellä. Uudessa sovelluksessa pyritään hyödyntämään paremmin verkkokaupan tuotetietoja ja tuotekuvia. Asiakas voi lukea tuotteen tuotekuvauksen ja nähdä, miltä tuote näyttää. Muuten toiminnot vastaavat alkuperäistä sovellusta.

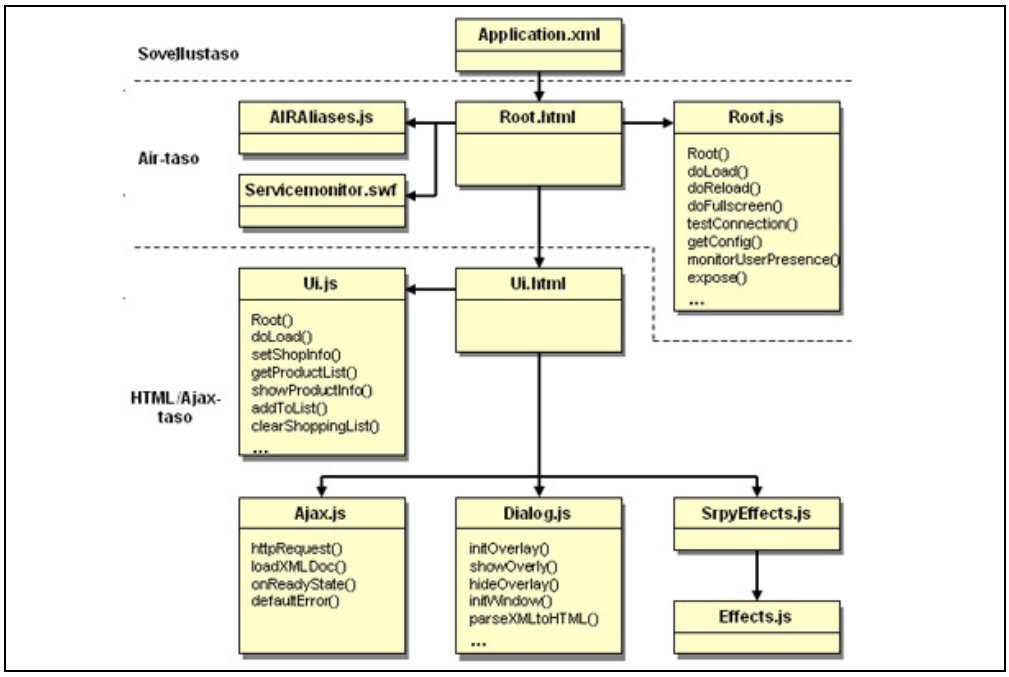
6.1.1 Rakenne

Järjestelmä koostuu kolmesta erillisestä osasta. Asiakaspäätteessä toimivasta Air-tekniikalla toteutetusta Asiakasselain-sovelluksesta, tavaratalon palvelinsovelluksesta ja verkkokaupan palvelinsovelluksesta. Sovelluksen rakenne on kuvattu järjestelmätasolla alla olevassa kuvassa (Kuva 10). Asiakasselain kommunikoi palvelinsovellusten kanssa HTTP-protokollalla. Asiakasselain hakee tavaratalon omasta varasto-tietokannasta tuotteen hinnan, saldon ja hyllypaikan. Muut tuotetiedot ja tuotekuvat haetaan verkkokaupan tietokannasta. Air-sovelluksella voidaan hakea tietoa useasta eri lähteestä, mikä mahdollistaa monipuolisten sovellusten toteuttamisen.



Kuva 10. Sovelluksen rakenne järjestelmätasolla

Air-sovellus koostuu aina vähintään kahdesta tiedostosta: sovelluksen kuvaustiedostosta sekä varsinaisesta sovellustiedostosta. Asiakasselain-sovellus on rakenteeltaan monimutkaisempi ja sisältää useita tiedostoja, jotka sisältävät eri tyyppisiä toimintoja. Seuraavassa on pyritty esittämään sovelluksen rakenne eri tiedostojen suhteen. Kaavio ei vastaa mitään erityistä UML-mallia, vaan pyrkii havainnollistamaan sovelluksen rakennetta graafisesti (Kuva 11).



Kuva 11. Sovelluksen rakenne tiedostotasolla

Sovelluksen rakenne voidaan pitää kolmitasoisena sen mukaan, mitä toimintoja kyseisellä tasolla toteutetaan. Ylimmällä, sovellustasolla, on sovelluksen kuvaustiedosto *application.xml*. Kuvaustiedostossa on määritelty sovelluksen ominaisuudet, sekä sovelluksen varsinainen sisältötiedosto *root.html*. Air-tasolla olevat tiedosto liittyvät Air-toimintojen toteuttamiseen. Alimmalla, HTML/Ajax-tasolla, ovat ne tiedostot, jotka toteuttavat sovelluksen käyttöliittymän ja palvelinkommunikoinnin. Seuraavassa on esitetty lyhyesti eri tiedostojen merkitys sovelluksessa.

Application.xml

Application.xml -kuvaustiedosto sisältää Air-sovelluksen määritykset. Sen avulla määritetään tarkasti sovelluksen ominaisuudet. Kuvaustiedosto on pakollinen kaikissa Air-sovelluksissa.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<application xmlns="http://ns.adobe.com/air/application/1.0">
    <id>fi.tuomas.StoreBrowser</id>
    <filename>StoreBrowser</filename>
    <name>StoreBrowser</name>
    <version>1.0</version>
    <copyright>Tuomas Haimi 2008</copyright>
    <initialWindow>
        <content>root.html</content>
        <title>Asiakasselain</title>
        <systemChrome>standard</systemChrome>
        <transparent>false</transparent>
        <visible>true</visible>
        <minimizable>true</minimizable>
        <maximizable>true</maximizable>
        <resizable>true</resizable>
        <width>1024</width>
        <height>768</height>
    </initialWindow>
    ...
</application>
```

Yllä on esitetty osa Asiakasselain-sovelluksen kuvaustiedostosta. Ominaisuuksien määrittely kuvaustiedoston avulla on selkeää ja yksinkertaista. Osaa määrityksistä voidaan muokata ajonaikaisesti sovelluksesta käsin. Kuvaustiedosto on esitetty kokonaisuudessaan liitteessä 1.

Root.html ja root.js

Root.html on Asiakasselain-sovelluksen varsinainen sovellustiedosto, joka on määritetty kuvaustiedoston `content`-elementillä. Tiedosto ladataan sovelluksen sisällöksi käynnistyksen yhteydessä. *Root.html*–tiedosto on esitetty liitteessä 2. Asiakasselain-sovellus koostuu kolmitasoisesta rakenteesta, jonka avulla erotetaan Air- ja HTML/Ajax-toiminnot toisistaan. Toimintojen erottamisella parannetaan erityisesti sovelluksen tietoturvaa, jota käsitellään tarkemmin luvussa 6.2.1.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Asiakasselain</title>
    ...
    <script type="text/javascript" src="js/root.js"></script>
  </head>
  <body>
    <div id="uidiv">
      <iframe id="ui"
         src="ui.html"
         sandboxRoot="http://biltemastorebrowser.fi/"
         documentRoot="app:/ui/"
         allowcrossDomainxhr="true"
         width="100%"
         height="100%"
         scrolling="yes">
      </iframe>
    </div>
  </body>
</html>
<!-- Application init -->
<script>
   var root = new Root();
   root.doLoad();
</script>
```

Kaikki sovelluksen Air-toiminnot suoritetaan `application`-hiekkalaatikossa ja HTML/Ajax-toiminnot suoritetaan `iframe`-elementtin ladattavassa erillisessä tiedostossa, joka puolestaan kuuluu `non-application`-hiekkalaatikkon. Sovelluksen toiminta käynnistyy `root.doLoad()`-komennolla.

Root.js on JavaScript-tiedosto, johon on keskitetty kaikki Air-toimintojen toteutukset. Toimintojen keskittäminen yhteen tiedostoon helpottaa sovelluksen ylläpitoa ja lisää tietoturvaa. Asiakasselain-sovelluksessa Air-toimintoina on tiedostojenkäsittely, verkkoyhteyksien ja sovelluksen tilan tarkkailu.

```
/* root.js */
var configFile;
var configXML;
var stream;
var monitor;
var status;
function Root(){
  this.doLoad = function(){
    this.doFullscreen();
    this.testConnection();
    this.expose();
    this.monitorUserPresence();
  }
  ...
}
```

Root.js ladataan *Root.html*-tiedostossa, joten se kuuluu `application`-hiekkalaatikkoon ja sillä on täydet oikeudet käsitellä Air-rajapintoja ja tietokoneen resursseja. *Root.js* -tiedosto on esitetty kokonaisuudessaan liitteessä 3.

AIRAliases.js ja *Servicemonitor.swf*

AIRAliases.js–tiedosto on SDK:n mukana toimitettava aputiedosto. Se sisältää hyödylliset lyhenteet Air-rajapintojen hyödyntämiseen JavaScriptillä. Esimerkiksi tiedostojenkäsittelyyn käytettävät kutsut ovat muotoa `window.runtime.flash.filesystem.FileStream`, jotka voidaan lyhentää muotoon `air.FileStream`. Lyhenteiden käyttö nopeuttaa ohjelmointia ja selkeyttää syntyvää ohjelmakoodia huomattavasti. Tiedosto liitetään sovellukseen *Root.html*–tiedoston `head`-osassa käyttämällä `script`-elementtiä.

```
/* root.html */
...
<script type="text/javascript" src="js/AIRAliases.js">
</script>
...
```

Servicemonitor.swf on flash-tiedosto, jonka avulla Air-sovelluksessa voidaan käyttää verkkoyhteyksien valvontaa. Verkkoyhteyksien valvonta on esitetty tarkemmin luvussa 6.2.3.

SpryEffects.js ja *Effects.js*

Sovelluksen visuaaliset tehosteet toteutetaan Adobe Spry Ajax -kirjastoon kuuluvilla *SpryEffects.js*- ja *Effects.js*-tiedostoilla. Tehosteiden lisääminen sovellukseen on esitetty tarkemmin luvussa 6.3.4.

Ui.html ja Ui.js

Ui.html sisältää Asiakasselain-sovelluksen HTML-kielellä toteutetun käyttöliittymän. Käyttöliittymän tapahtumankäsittelijät liitetään kutsumalla *Ui.js*-tiedoston `doLoad()`-funktiota. *Ui.html*-tiedosto on esitetty kokonaisuudessaan liitteessä 4.

Ui.js-tiedostoon on koottua kaikki sovelluksen käyttöliittymän muokkaus toiminnot, tapahtumankäsittely ja tarvittavat apufunktiot. Ajax-toiminnot esitellään tarkemmin luvussa 6.3 ja *Ui.js*-tiedosto on esitetty kokonaisuudessaan liitteessä 5.

Dialog.js

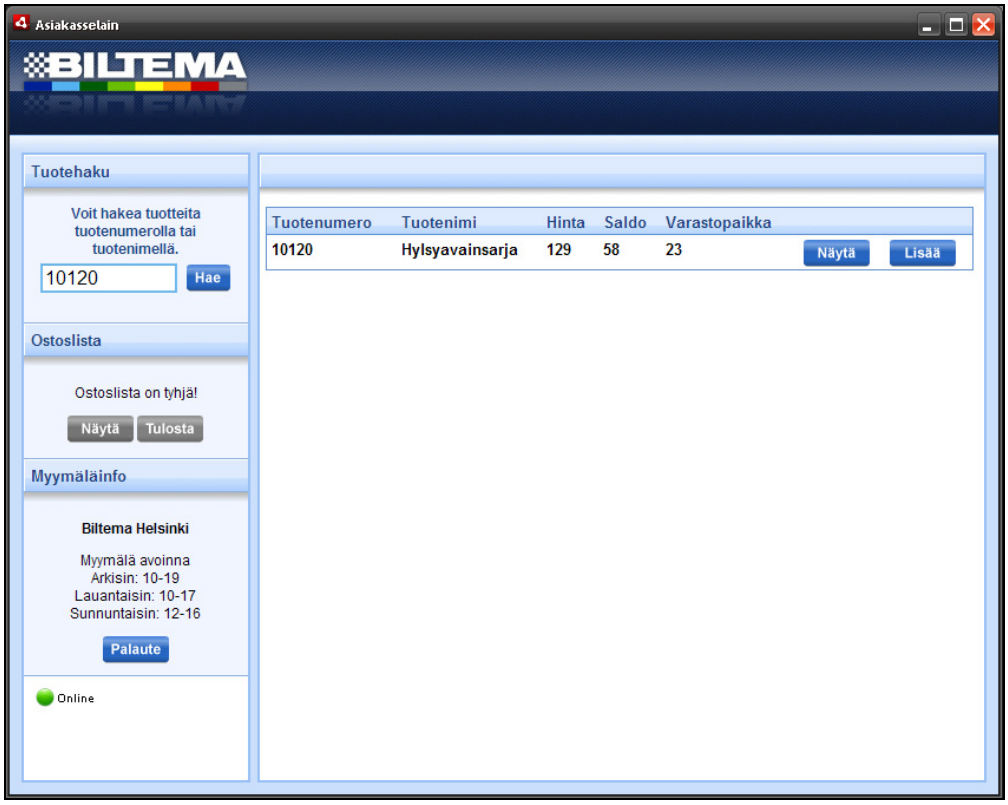
Dialog.js sisältää JavaScript-ohjelmakoodia, jonka avulla luodaan Asiakasselain-sovelluksen modaalisen ponnahdusikkunat. Ikkunoissa näytetään tuotteen tiedot ja kuva, sekä ostolista. *Dialog.js*-tiedosto on esitetty kokonaisuudessaan liitteessä 7.

Palvelinsovellus

Sovelluksen kehitysvaiheessa ei toteutettu varsinaista palvelinsovellusta, joka toimii tavaratalon palvelimella ja hyödyntää oikeaa varasto-tietokantaa. Kehitystyötä varten ohjelmoitiin yksinkertainen PHP-palvelinsovellus, joka käyttää MySql-tietokantaa tuotetietojen tallentamiseen. Palvelinsovelluksen tehtävänä on vastaanottaa Asiakasselain-sovelluksen lähettämät pyynnöt, suorittaa tarvittavat tietokantahaut ja palauttaa vastaus XML-muodossa takaisin Asiakasselain-sovelluksella.

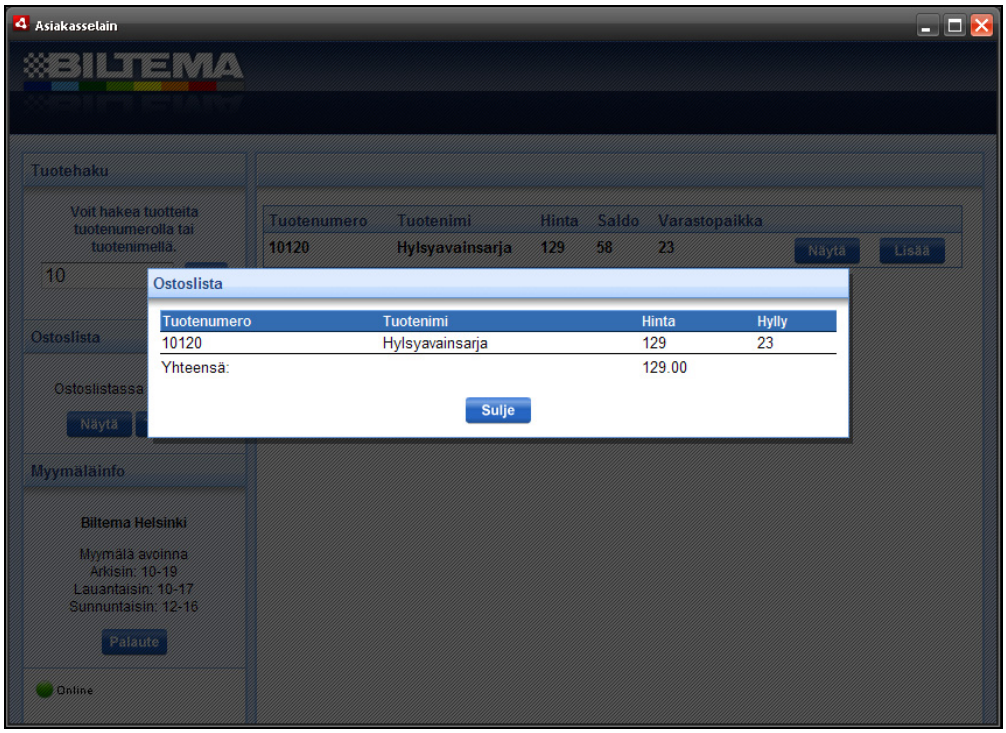
6.1.2 Käyttöliittymä

Asiakasselaimen käyttöliittymä on toteutettu HTML-kielellä ja CSS-tyylitiedostoilla (Kuva 12). Käyttöliittymän toiminta ja käyttäjän toimiin reagoiminen perustuu JavaScript-ohjelmointikieleen ja elementtien dynaamiseen muokkaamiseen. Käyttöliittymä koostuu yhdestä HTML-tiedostosta ja on tyypiltään yhden sivun käyttöliittymä, jota muokataan toimintojen mukaan.



Kuva 12. Asiakasselaimen käyttöliittymä

Käyttöliittymän vasen puoli sisältää sovelluksen vakiotoiminnot, joista ylimpänä on tuotehaku, seuraavana ostoslistan hallintapainikkeet ja alimpana tavaratalon info-ruutu. Oikea puoli käyttöliittymästä on varattu hakutulosten esittämiseen.



Kuva 13. Asiakasselain-käyttöliittymän ponnahdusikkuna

Tuotetietojen ja ostolistan näyttämiseen sekä asiakaspalautteen antamiseen käytetään modaalisia ponnahdusikkunoita, jotka erottuvat selkeästi muusta käyttöliittymästä (Kuva 13). Käyttöliittymä on pyritty pitämään selkeänä ilman ylimääräisiä toimintoja. Tyylikkyyttä on lisätty käyttämällä pieniä visuaalisia tehosteita ja yritysilmeen mukaista ulkoasua ja väritystä. Käyttöliittymän ulkoasua voidaan muokata helposti CSS-tyyliohjeiden avulla.

6.1.3 Toiminta

Asiakasselain-sovellus on suunniteltu toimimaan normaalilla Pc-laitteistokokoonpanolla. Sovellusta ohjataan hiiren ja näppäimistön avulla ja lisäksi käytössä on kuittikirjoitin ostoslistan tulostamista varten. Käyttäjä voi hakea tuotteita tuotenumeron tai tuotenimen perusteella. Hakutulokset näytetään listana, josta käyttäjä voi lisätä tuotteen ostoslistaan tai lukea lisätietoja tuotteesta. Löydettyään haluamansa tuotteet käyttäjä voi tulostaa tai poistaa ostoslistan. Sovelluksella voi myös antaa asiakaspalautetta tavaratalon henkilökunnalle. Asiakasselain-sovellus sisältää toiminnon, joka palauttaa sovelluksen alkutilaan oltuaan määritellyn ajan käyttämättä.

6.2 Air-toiminnot

Asiakasselain-sovelluksessa hyödynnetään osaa Air-rajapintojen tarjoamista toiminnoista, joita perinteisissä web-sovelluksissa ei ole mahdollista käyttää. Tärkeimpinä toimintoina on tietoturvan toteutus, tiedostojenkäsittely sekä verkkoyhteyksien valvonta. Air-sovelluksessa voidaan käyttää myös muita edistyksellisiä toimintoja, mutta niiden hyödyntäminen kyseisessä sovelluksessa ei tuo mainittavaa lisäarvoa.

6.2.1 Tietoturva

Air-sovelluksen tietoturva perustuu Internet-selaimista tuttuun hiekkalaatikkomalliin, jossa mahdollisesti vaarallista ohjelmakoodia sisältävät osat suoritetaan erillään muusta järjestelmästä. Hiekkalaatikot on jaettu kahteen osaan: `application`- ja `non-application` -hiekkalaatikkoon. Asiakasselain-sovelluksessa kaikki Air-rajapintoja ja tietokoneen resursseja hyödyntävät toiminnot on keskitetty yhteen tiedostoon ja ne suoritetaan `application`-hiekkalaatikossa. Kaikki muu toiminta, kuten palvelinsovelluksen kanssa kommunikointi, suoritetaan `non-application` -hiekkalaatikossa. Eristämällä toiminnot toisistaan voidaan estää haitallisen koodin pääsy tietokoneen resursseihin ja esimerkiksi tuhoamaan tiedostoja.

Asiakasselain koostuu kahdesta HTML-tiedostosta: `root.html` -tiedosto on sovelluksen varsinainen sovellustiedosto, joka on määritelty kuvaustiedoston `content`-elementillä. Se ladataan ensimmäisenä sovelluksen käynnistämisen yhteydessä. Tiedosto kuuluu automaattisesti `application`-hiekkalaatikkoon. Sovelluksen käyttöliittymä, `ui.html` ladataan `root.html`-tiedostossa määriteltyyn `iframe`-elementtiin.

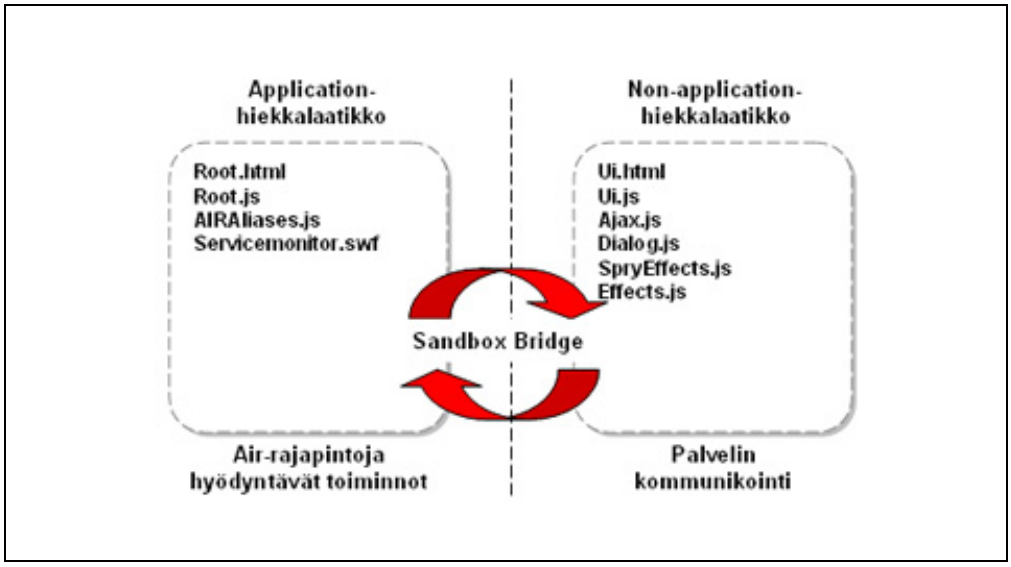
```
...
<iframe id="ui"
    src="ui.html"
    sandboxRoot="http://biltemastorebrowser.fi/"
    documentRoot="app:/ui/"
    allowcrossDomainxhr="true"
    width="100%"
    height="100%"
    scrolling="yes">
</iframe>
...
```

Iframe-elementin `sandboxRoot` -ominaisuudelle määritetään kuvitteellinen osoite, jolloin näyttää siltä, että tiedosto ladataan toisesta verkko-osoitteesta. Käytännössä `ui.html`-tiedosto ladataan `documentRoot` -ominaisuuden

ilmaisemasta lähteestä, joka tässä viittaa suoraan sovelluksen asennuskansion `ui`-kansioon. Näin määriteltynä `ui.html` -tiedosto kuuluu `non-application` -hiekkalaatikkoon ja sillä on rajoitetut oikeudet hyödyntää Air-rajapintoja ja tietokoneen resursseja. Eri hiekkalaatikoihin kuuluvat tiedostot kannattaa järjestää omiin hakemistoihin.

Hiekkalaatikoiden yhteistoiminta

Monipuolisesti toimivan sovelluksen toteuttamiseksi tarvitaan kuitenkin menetelmä, jonka avulla eri hiekkalaatikoihin kuuluvat sovelluksen osat voivat käyttää Air-rajapintojen tarjoamia palveluita. Air sisältää `Sandbox Bridge` -nimisen toiminnon, jonka välityksellä `application`- ja `non-application` -hiekkalaatikot voivat kommunikoida keskenään (Kuva 14). Sovelluskehittäjä voi yksikäsitteisesti sallia ne toiminnot, joita voidaan käyttää `non-application`-hiekkalaatikosta käsin.



Kuva 14. SandboxBridge ja hiekkalaatikoiden yhteistoiminta

`SandboxBridge` otetaan käyttöön `root.js` tiedostossa, joka kuuluu `application`-hiekkalaatikkoon. Funktiot, joita halutaan kutsua sovelluksen `non-application`-hiekkalaatikkoon kuuluvista osista, määritetään yhden olion funktioiksi.

```
/* root.js */
...
this.expose = function(){
    var Exposed = {};
    Exposed.getConfig = this.getConfig;
    Exposed.doReload = this.doReload;
    Exposed.connectionStatus = this.connectionStatus;
    docment.getElementById('ui').
     contentWindow.parentSandboxBridge = Exposed;
}
...
```

Olio alistetaan käyttöön `iframe`-elementin `parentSandboxBridge`-ominaisuudella. `Non-application`-hiekkalaatikkoon kuuluva tiedosto voi kutsua alistetun olion funktioita `parentSandboxBridge`-ominaisuuden kautta.

```
/* ui.js */
...
/* load config via sandboxbridge */
settings = parentSandboxBridge.getConfig();
...
```

6.2.2 Tiedostojen käsittely

Air sisältää monipuoliset toiminnot tiedostojenkäsittelyyn. Tiedostoja ja kansioita voidaan lukea, muokata ja poistaa sekä luoda uusia. Asiakasselain hyödyntää tiedostojenlukumahdollisuutta, jonka avulla ladataan kyseisen asiakaspäätteen asetukset sovelluksen käynnistyksen yhteydessä. Asetukset on tallennettu XML-tiedostoon, jonka lukeminen on yksinkertaista ja nopeaa. Alla on esitetty sovelluksessa käytettävä asetustiedosto, `settings.xml`.

```
<?xml version="1.0" encoding="utf-8"?>
<settings>
   <storebrowser>
     <serverurl>http://localhost/sb/server.php</serverurl>
     <localurl></localurl>
     <imageurl>http://www.biltema.fi/data/kuvat/</imageurl>
     <browserid>Browser 1</browserid>
     <defaultlang>fi</defaultlang>
    </storebrowser>
    <shopinfo>
      <name>Biltema Helsinki</name>
      <businesshours workdays="10-19"
                     saturday="10-17"
                     sunday="12-16">
      </businesshours>
    </shopinfo>
</settings>
```

Asetustiedoston lukemiseen käytettään Air-ajoympäristön *File I/O*-rajapintaa, joka tarjoaa käyttöjärjestelmäriippumattomat metodit tiedostojenkäsittelyyn.

```
/**
 * function getConfig()
 * - reads config.xml file
 */
this.getConfig = function(){
   configFile = air.File.documentsDirectory.
     resolvePath("settings.xml");
   stream = new air.FileStream();
   if (configFile.exists) {
      stream.open(configFile, air.FileMode.READ);
      configXML = stream.readUTFBytes(stream.bytesAvailable);
       stream.close();
      ...
   }
   else{
      return "Asetustiedostoja ei löydy!";
   }
}
```

Asetustiedosto tallennetaan tietokoneen *Omat tiedostot* -kansioon, johon voidaan viitata `air.File.documentsDirectory.resolvePath`-funktiolla. Tiedoston lukua varten avataan tavuvirta. Seuraavaksi tarkistetaan, onko tiedosto olemassa ja avataan se lukemista varten. Tiedoston lukeminen on asynkroninen operaatio, joten se ei keskeytä sovelluksen toimintaa. Tiedosto luetaan tavuina muistiin, jonka jälkeen tavuvirta suljetaan ja tiedosto voidaan parsia XML-tiedostoksi ja edelleen muuttaa JavaScript-olioksi.

6.2.3 Verkkoyhteydet

Asiakasselain-sovelluksessa käytetään Air-ajoympäristön tarjoamaa verkkoyhteyden valvontatoimintoa. Sen tehtävänä on tarkkailla yhteyksiä palvelimeen ja yhteyden ollessa poikki estää pyyntöjen lähettäminen ja näyttää virheilmoitus käyttäjälle. Air SDK:n mukana tulee erillinen `servicemonitor.swf`-tiedosto, joka sisällytetään sovellukseen HTML-tiedostossa.

```
/* root.html */
...
<script src="servicemonitor.swf" type="application/x-
shockwave-flash"/>
...
```

Air-ajoympäristö tarkkailee verkkoyhteyksiä, ja verkkoyhteyden tilan muuttuessa se luo uuden `StatusEvent.STATUS`-tapahtuma, joiden kuuntelemiseen asetetaan tapahtumankuuntelija. Tapahtumankuuntelijan avulla verkkoyhteyden muutoksiin voidaan reagoida halutulla tavalla. Asiakasselaimes-

sa käytetään `URLMonitor`-toimintoa, joka tarkkailee HTTP-yhteyden tilaa palvelimen osoitteeseen.

```
/* root.js */
...
/**
 * function testConnection()
 */
this.testConnection = function(){
  var url = new air.URLRequest
     ('http://localhost/storebrowser/');
  monitor = new air.URLMonitor(url);
  monitor.addEventListener(air.StatusEvent.STATUS,
     this.connectionStatus);
  monitor.pollInterval = 2000;
  monitor.start();
}

/**
 * function connectionStatus()
 */
this.connectionStatus = function(e){
  if(monitor.available){
    status = 1;
  }
  else{
    status = 0;
  }
  return status;
}
...
```

Verkkoyhteyttä valvovalle monitorille asetetaan tapahtumankuuntelija, joka tarkkaillee verkkoyhteyden tilaa, ja muutoksen tapahtuessa kutsuu haluttua funktiota. Lisäksi monitorille voidaan antaa millisekunteina tarkasteluväli, kuinka usein monitori tarkastaa verkkoyhteyden tilan. Monitori alkaa tarkkaila verkkoyhteyden tilaa vasta, kun se käynnistetään `monitor.start()` -komennolla.

6.2.4 Sovelluksentilan tarkkailu

Asiakasselain-sovelluksessa käytetään sovelluksentilan tarkkailua, jonka avulla päätellään kauanko sovellus on ollut käyttämättä. Ominaisuutta käytetään tilanteessa, jossa asiakas on selaillut tuotteita ja lähtenyt pois päätteen luota. Kun sovellus on ollut käyttämättä määrätyn ajan, päivitetään sovellus takaisin alkutilaan.

```
/**
 * function userPresence()
 */
this.monitorUserPresence = function(){
  air.NativeApplication.nativeApplication.idleThreshold = 120;
  air.NativeApplication.nativeApplication.addEventListener
    (air.Event.USER_IDLE, this.onIdle);
  air.NativeApplication.nativeApplication.addEventListener
    (air.Event.USER_PRESENT, this.onPresence);
}

/**
 * function onIdle()
 */
this.onIdle = function(){
  var ui = document.getElementById('ui');
  ui.src = "ui.html";
}

/**
 * function onPresence()
 */
this.onPresence = function(){
  // DUMMY_FUNCTION
}
```

Air-ajoympäristö tarkkailee hiiri- ja näppäimistötoimintoja ja tilanteen mukaan luo `Event.USER_IDLE`- tai `Event.USER_PRESENT` -tapahtuman. Kun sovellus on ollut käyttämättä määrätyn ajan, kutsutaan `onIdle()`-funktiota, joka päivittää sovelluksen takaisin alkutilaan.

6.3 Ajax-toiminnot

Ajax-toiminnoiksi lasketaan suurin osa Asiakasselain-sovelluksen JavaScript-ohjelmointiin perustuvista toiminnoista. Tärkeimpiä toimintoja on tapahtumankäsittely, palvelinkommunikointi ja käyttöliittymän dynaaminen muokkaus sekä tiedon esittäminen käyttäjälle.

6.3.1 Tapahtumankäsittely

Sovelluksen graafinen käyttöliittymä generoi tapahtumia. Yleisesti graafista käyttöliittymää hyödyntävää sovellusta kutsutaan tapahtumaohjatuksi sovellukseksi. HTML/Ajax-tekniikalla toteutetussa Air-sovelluksessa HTML-elementteihin lisätään tapahtumankäsittelijöitä, jotka reagoivat erilaisiin tapahtumiin. Tapahtumat voivat olla käyttäjän tai järjestelmän tuottamia. Käyttäjä tuottaa tapahtumia käyttämällä hiirtä tai näppäimistöä. Järjestelmä tuottaa erilaisia tapahtumia erityisesti järjestelmän tilan muutosten yhteydessä. [33, s. 110.]

Asiakasselain-sovelluksessa käytetään pääsääntöisesti kahta tapahtumankäsittelymallia. Air-toimintoihin liittyvät tapahtumankäsittelijät voidaan rekisteröidä ainoastaan W3C:n Document Object Model Level 3 Event Specification suositusten mukaisella tavalla. Tapahtumankäsittelijät lisätään `addEventListener()`-funktiolla. Parametreina annetaan merkkijonona tapahtuma, jota kuunnellaan ja toisena parametrina kutsuttavan funktion nimi.

```
/**
 * function testConnection()
 */
this.testConnection = function(){
  var url = new air.URLRequest
    ('http://localhost/storebrowser/');
  monitor = new air.URLMonitor(url);
  monitor.addEventListener
    (air.StatusEvent.STATUS, this.connectionStatus);
  monitor.pollInterval = 2000;
  monitor.start();
}
```

Ajax-toimintojen yhteydessä käytetään ns. perinteistä tapahtumankäsittelymallia. HTML-elementeille, joihin tapahtumankäsittelijä liitetään, määritetään yksikäsitteinen tunniste, *id*. Tunnisteen avulla kyseiseen elementtiin voidaan viitata JavaScript-koodissa. Kutsuttava funktio annetaan jonkin tapahtumankäsittelijän ominaisuudeksi. Käyttämällä anonyymejä funktioita voidaan yhdelle tapahtumankäsittelijälle ohjelmoida monimutkaisia toimintoja.

```
...
/* add eventlistener to feedback-button */
feedback_btn.onclick = function(){
  var _url = settings.serverurl+"?action=feedback";
  var req = new Ajax.httpRequest(_url, showFeedback, true);
}
...
```

HTML/Ajax-tekniikalla toteutettu Air-sovellus ei poikkea tapahtumankäsittelyltään perinteisestä web-sovelluksesta. Mahdollisuus käyttää erilaisia malleja tekee sovelluksen suunnittelusta ja toteuttamisesta joustavampaa.

6.3.2 Palvelinkommunikointi

Asiakasselain-sovelluksen ja palvelimen välinen kommunikointi toteutetaan XMLHttpRequest-olion asynkronisilla kutsuilla. Palvelin kommunikointiin tarvittavat funktiot on koottu erilliseen JavaScript-tiedostoon, joka on esitetty kokonaisuudessaan liitteessä 6. Air-ajoympäristö tukee kommunikointia eri verkkoalueiden välillä, mikä ei ole sallittua Internet-selaimissa. Toiminto on kuitenkin oletusarvoisesti estetty, ja se pitää sallia erikseen ennen käyttöä.

```
/* root.html */
...
<div id="uidiv">
  <iframe id="ui"
    src="ui.html"
    sandboxRoot="http://biltemastorebrowser.fi/"
    documentRoot="app:/ui/"
    allowcrossDomainxhr="true"
    width="100%"
    height="100%"
    scrolling="yes">
  </iframe>
</div>
...
```

Palvelimelle lähetetään XMLHttpRequest-pyyntö, jolle annetaan parametrit URL-muodossa sekä viite vastauksen käsittelevään funktioon.

```
/* ui.js */
...
/**
 * function getProductList()
 */
function getProductList(){
  var search_str = $('search_str');
  if(search_str.value == ""){
    showPopup("Anna tuotenumero tai tuotenimi!");
  }
  else{
    var _args = "";
    var form = $('search_form');
    var form_length = form.elements.length;
    for(var i = 0; i < form_length; i++){
      if(i < form_length - 1){
      _args += form.elements[i].name+"="
      +encodeURIComponent(form.elements[i].value)+"&";
    }
    else{
      _args += form.elements[i].name+"="
      +encodeURIComponent(form.elements[i].value);
    }
  }
    var _url = settings.serverurl+"?"+_args;
    var req = new Ajax.httpRequest(_url, showProductList);
  }
}
...
```

Palvelin vastaanottaa ja käsittelee pyynnön sekä suorittaa tarvittavat tietokantahaut. Palvelin muodostaa haun tuloksista XML-muotoisen vastauksen ja lähettää sen takaisin Asiakasselain-sovellukselle, joka käsittelee vastauksen ja näyttää tulokset käyttäjälle.

```
/* ui.js */
/**
 * function showProductInfo()
 */
function showProductInfo(){
  var xml = this.req.responseXML;
  initWindow(xml, parseProductXML, productDialog,
    "Tuotetiedot");
}
...
```

Asynkroninen kommunikointi ei keskeytä sovelluksen muuta toimintaa, mikä tekee sovelluksen käyttämisestä mielekästä. Lisäksi käyttäjälle näytetään visuaalinen viesti siitä, että sovellus käsittelee parhaillaan käyttäjän pyyntöä.

6.3.3 Käyttöliittymän dynaaminen muokkaus

Palvelimen lähettämä XML-muotoinen vastaus muokataan HTML-merkkaukseksi ja näytetään käyttäjälle. Muokkaus voidaan toteuttaa kahdella tavalla. Standardin mukainen menetelmä on käyttää DOM-metodeja, joilla luodaan kaikki elementit yksitellen. DOM-metodeja käyttämällä HTML-merkkauksen luonti on monimutkaista ja yleensä hitaampi vaihtoehto. Asiakasselain-sovelluksessa HTML-merkkaus luodaan merkkijonona ja liitetään HTML-dokumenttiin halutun elementin sisällöksi `innerHTML` -ominaisuuden avulla.

```
/* ui.js */
/**
 * function showProductList()
 */
function showProductList(){
  var div = $('productlist');
  div.style.display = 'none';
  var html = parseProductXml(this.req.responseXML);
  div.innerHTML = html;
  ...
}
```

XML-muotoisen tiedon muokkaaminen HTML-merkkaukseksi on helppo toteuttaa silmukassa.

```
/* ui.js */
/**
 * function parseProductXml(xmldoc)
 */
function parseProductXml(xmldoc){
  var number = xmldoc.getElementsByTagName('productnumber');
  var name = xmldoc.getElementsByTagName('productname');
  var price = xmldoc.getElementsByTagName('productprice');
  var saldo = xmldoc.getElementsByTagName('productsaldo');
  var place = xmldoc.getElementsByTagName('productplace');
  var html;
  if(number.length == 0){
    html = "<h3>Hakemaasi tuotetta ei löytynyt</h3>"
  }
  else{
   html = "<div class=\"test\">
           <table>
            <tr>
             <td Tuotenumero</td>
             <td Tuotenimi</td>
             <td>Hinta</td>
             <td>Saldo</td>
             <td>Varastopaikka</td>
             <td></td><td></td>
            </tr>";
   for(var i = 0; i < number.length; i++){
     html += "<tr id=\"tr"+i+"\" >";
     html += "<td>"+number[i].firstChild.nodeValue+"</td>
              <td>"+name[i].firstChild.nodeValue+"</td>";
     html += "<td>"+price[i].firstChild.nodeValue+"</td>
              <td>"+saldo[i].firstChild.nodeValue+"</td>
              <td>"+place[i].firstChild.nodeValue+"</td>";
     html += "<td><img src=\"btn_show_up.png\"
               id=\"g"+number[i].
               firstChild.nodeValue+"\" /></td>";
     html += "<td><img src=\"btn_add_up.png\"
               id=\"a"+number[i].
               firstChild.nodeValue+"\" /></td>";
     html += "</tr>";
   }
   html += "</table></div>";
   }
  return html;
}
```

6.3.4 *Visuaaliset tehosteet*

Rikkaat Internet-sovellukset sisältävät useasti erilaisia visuaalisia tehosteita, joiden avulla elävöitetään käyttöliittymää ja lisätään sovelluksen näyttävyyttä. Asiakasselain-sovelluksessa käytetään tehosteita tilanteissa, joissa käyttöliittymää muokataan dynaamisesti. Tehosteiden avulla korostetaan, että sovelluksen sisältöä on muutettu. Esimerkiksi käyttäjän lisätessä tuotteita ostoslistaan, korostetaan ostoslistan taustaväri keltaiseksi ja häivytetään takaisin normaaliin väriin (Kuva 15).

Kuva 15. Esimerkki visuaalisesta tehosteesta

Tehosteet on toteutettu Adobe Spry Ajax -kirjaston avulla. Tehosteiden toteuttamiseen on tarjolla hyvin paljon erilaisia ratkaisuja. Adobe valintaan vaikutti etenkin tehosteiden lisäämisen helppous ja yhteensopivuus Air-tekniikan kanssa.

```
/* ui.js*/
/**
 * function updateProductlist()
 */
 function updateProductlist(){
 ...
   Spry.Effect.DoHighlight('plistdiv',
     {duration:1000,from:'#ffff99',to:'#eff5ff'});
 ...
}
```

Tehosteiden lisäämiseen on yksinkertaiset funktiot. Parametrina annetaan elementin tunniste, johon tehoste kohdistetaan sekä tehosteen kestoaika ja tarvittavat väriarvot heksadesimaalimuodossa.

6.4 Kokemukset Adobe Air -sovelluksesta Ajax-tekniikalla

Adobe Air vaikuttaa erittäin mielenkiintoiselta ja lupaavalta vaihtoehdolta työpöytäsovelluksen toteuttamiseen. Adoben tuotteistus on onnistunut hyvin ja tekniikka vaikuttaa valmiilta ja toimivalta. Yksinkertaisen sovelluksen toteuttaminen vaatii ainoastaan perustiedot Air-tekniikasta ja käytettävästä sovellustekniikasta. Air sisältää paljon ominaisuuksia, joita ei kuitenkaan insinöörityön aikana tarvittu. Erityisesti yllätti ohjelmointityön helppous ja nopeus. Osittain se johtuu HTML/Ajax-tekniikan tuntemuksesta. HTML/Ajax-tekniikka on toimiva valinta sovellustekniikaksi, koska Air tukee hyvin eri Ajax-kirjastoja ja XMLHttpRequest-oliota. Työn aikana ilmeni kuitenkin eräitä piirteitä, jotka vaikuttavat mielipiteeseen Air-tekniikasta. Suurimpana ongelmana Asiakasselain-sovelluksen kohdalla on, ettei Air-ajoympäristö tue ainakaan vielä ”kioskitilaa”, joka on julkisessa tilassa käytettävän sovelluksen perusedellytys. Kyseinen ominaisuus on varmasti toteutettavissa kiertoteitse, mutta ongelmaan ei vielä löytynyt toimivaa ratkaisua. Toinen harmittava ongelma liittyy dokumentaatioon. Adobe tarjoaa ilmaiseksi monipuolisia oppaita ja esimerkkisovelluksia. Materiaalissa esiintyy kuitenkin häiritseviä virheitä ja dokumentoimattomia ominaisuuksia, joita selvitellessä kului paljon aikaa.

Ajan puutteen takia Asiakasselain-sovellusta ei vielä päästy testaamaan aidossa käyttöympäristössä. Seuraavana vaiheena onkin tehostaa ja selkeyttää ohjelmakoodia ja korjata mahdollisia virheitä. Uuden version myötä sovellusta voitaneen jo testata aidossa käyttöympäristössä.

7 YHTEENVETO

Insinöörityön päätavoitteena oli tutustua Adobe Air-tekniikkaan ja sen tarjoamiin mahdollisuuksiin työpöytäsovelluksen toteuttamisessa. Työn teoriaosuudessa käytiin lyhyesti läpi web-sovelluksen kehitys rikkaaksi Internet-sovellukseksi, jonka jälkeen keskityttiin Air-tekniikan tutkimiseen. Teoriaosuuden ansiosta insinöörityön toinen päätavoite eli sovelluksen toteutus Air-tekniikalla voitiin täyttää.

Asiakasselain-sovelluksella on tarkoitus korvata tavarataloissa käytettävä asiakaspääte, jota asiakkaat käyttävät etsiessään tuotteita ja tuotetietoa. Air-tekniikka osoittautui erittäin helposti omaksuttavaksi ja vaikuttaa varteenotettavalta vaihtoehdolta työpöytäsovelluksien toteuttamiseen. Tulevat kuukaudet osoittavat kuinka laajan suosion Air-tekniikka saavuttaa sovelluskehittäjien parissa. Leviämistä nopeutta varmasti Air-tuen lisäämisen Linux-käyttöjärjestelmille ja mahdollisesti mobiililaitteille.

Insinöörityö onnistui täyttämään sille asetetut tavoitteet. Työn suurimpana antina on Air-tekniikan tuntemus ja sen tarjoamat mahdollisuudet. Asiakasselain-sovellus ei vielä täytä tarvittavia kriteereitä, jotta se voitaisiin ottaa päivittäiseen käyttöön tavarataloissa. Osittain syynä on Air-ajoympäristön tekniset rajoitteet ja osittain sovelluksen kehittämiseen käytettävän ajan puute. Työ kuitenkin osoittaa, että Adobe Air on tulevaisuudessa harkinnan arvoinen tekniikka sovelluskehityksessä.

VIITELUETTELO

[1] Connolly, Dan. *A little History of the World Wide Web*. [verkkodokumentti] 2000, päivitetty 13.6.2006 [viitattu 13.11.2007]. Saatavissa: http://www.w3.org/History.html.

[2] Hintikka, Kari A. *Web2.0 - Johdatus internetin uusiin liiketoiminta-mahdollisuuksiin.* TIEKE Tietoyhteiskunnan kehittämiskeskus ry:n julkaisu-sarja osa 28. Helsinki. TIEKE RY. 2007.

[3] Wikipedia. *Web application*.[verkkodokumentti] [Viitattu 13.11.2007]. Saata-vissa: http://en.wikipedia.org/wiki/Web_application.

[4] W3C. *The Extensible HyperText Markup Language (Second Edition).* [verk-kodokumentti.] 2000, päivitetty 2002 [viitattu 13.11.2007]. Saatavissa: http://www.w3.org/TR/xhtml1/.

[5] W3C. *Document Object Model (DOM)* [verkkodokumentti] 1997, päivitetty 19.1.2005 [viitattu 13.11.2007]. Saatavissa: http://www.w3.org/DOM/.

[6] Wikipedia. *CSS*.[verkkodokumentti] [Viitattu 13.11.2007]. Saatavissa: http://en.wikipedia.org/wiki/CSS.

[7] Wikipedia. *JavaScript*.[verkkodokumentti] [Viitattu 13.11.2007]. Saatavissa: http://en.wikipedia.org/wiki/JavaScript/.

[8] W3C. HTTP - *Hypertext Transfer Protocol* [verkkodokumentti] 1996, päivitet-ty 27.2.2008 [viitattu 13.11.2007]. Saatavissa: http://www.w3.org/Protocols/.

[9] Garret, Jesse James. *Ajax: A new Approach to Web Applications* [verkkodokumentti]. 18.2.2005 [viitattu 23.11.2007]. Saatavissa: http://www.adaptivepath.com/publications/essays/archives/000385.php.

[10] Allaire, Jeremy, Macromedia Inc. *Macromedia Flash MX - A next generation rich client* [verkkodokumentti]. 2002 [viitattu 17.11.2007]. Saatavissa: http://download.macromedia.com/pub/flash/whitepapers/richclient.pdf.

[11] Klemetti, Kettunen. Samcom Oy. *Mikä on Web 2.0* [verkkodoumentti] 23.7.2008 [viitattu 17.11.2007]. Saatavissa: http://www.samcom.fi/Asiakkaat/Samcom-White-Papers.html.

[12] Crane, Dave - Pascarello, Eric. *Ajax In Action*. Greenwich: Manning Publica-tions Co. 2006 [viitattu 8.8.2007].

[13] Bozzon - Comai - Fraternal - Carughi. *Conceptual Modeling and Code Gen-eration for Rich Internet Applications, Proceedings of the 6th international conference on Web engineering*. 2006 ACM Press, (353-360).

[14] Mozilla. *XML User Interface Language (XUL)* [verkkodokumentti] 1998, päi-vitetty 20.1.2008 [viitattu 17.11.2007]. Saatavissa: http://www.mozilla.org/projects/xul/.

[15] Adobe. *Developing Adobe® AIR™ Applications with HTML and Ajax* [verkkodokumentti] 2008 [viitattu 8.1.2008]. Saatavissa: http://livedocs.adobe.com/air/1/devappshtml/dev_guide_html.pdf.

[16] Laszlo Systems. An *Open Architecture Framework for Advanced Ajax Applications* [verkkodokumentti] 2006 [viitattu 8.1.2008]. Saatavissa: http://www.openlaszlo.org/whitepaper/LaszloWhitePaper.pdf.

[17] Chambers - Dura - Hoyt. *Adobe® Integrated Runtime (Air) for JavaScript Developers - Pocket Guide* [verkkodokumentti] 2007 [viitattu 8.1.2008]. Saatavissa:http://download.macromedia.com/pub/labs/air/books/adobe_air_for_javascript_developers_062807.pdf.

[18] SQLite. *SQLite features* [verkkodokumentti] 2008 [viitattu 8.1.2008]. Saatavissa: http://www.sqlite.org/features.html.

[19] Adobe. *Developing Adobe® AIR™ Applications with Flex* [verkkodokumentti] 2008 [viitattu 11.1.2008]. Saatavissa: http://livedocs.adobe.com/air/1/devappsflex/dev_guide_flex.pdf.

[20] Kazoun, Chafic - Lott, Joey. *Programming Flex 2*. California, US: O'Reilly Media Inc. 2007 [viitattu 8.8.2007].

[21] Adobe. *Adobe® FlexDeveloper Guider* [verkkodokumentti] 2008 [viitattu 11.1.2008]. Saatavissa: http://livedocs.adobe.com/flex/3/devguide_flex3.pdf.

[22] Adobe. *Adobe® AIR™ 1.0 Security White Paper* [verkkodokumentti] 22.1.2008 [viitattu 2.2.2008]. Saatavissa: http://download.macromedia.com/pub/air/documentation/1/air_security.pdf.

[23] Adobe. *Adobe® AIR™ 1.0 HTML Security White Paper* [verkkodokumentti] 22.1.2008 [viitattu 2.2.2008]. Saatavissa: http://download.macromedia.com/pub/air/documentation/1/air_htmlsecurity.pdf.

[24] Adobe. *Adobe® Flash Player Security White Paper* [verkkodokumentti] 2006 [viitattu 2.2.2008]. Saatavissa: http://www.adobe.com/devnet/flashplayer/articles/flash_player_9_security.pdf.

[25] Adobe. *Browser vs. Desktop* [verkkodokumentti] 2008 [viitattu 15.2.2008]. Saatavissa: http://www.adobe.com/products/air/comparison/.

[26] Junkkaala, Jouni. Adobe avaa ilmatilaa. *Tietoviikko* 7.12.2007.

[27] Hopman, Alex. *The Story of XMLHTTP* [verkkodokumentti] päivitetty 31.1.2007 [viitattu 23.11.2007]. Saatavissa: http://www.alexhopmann.com/xmlhttp.htm.

[28] W3C. *The XMLHttpRequest Object* [verkkodokumentti] 2007 [viitattu 23.11.2007]. Saatavissa: http://www.w3.org/TR/XMLHttpRequest/.

[29] Mozilla Developer Center. *Prism* [verkkodokumentti] 2008, [viitattu 12.2.2008]. Saatavissa: http://developer.mozilla.org/en/docs/Prism.

[30] Kuittinen, Petri. Silverlight haastaa flashin. *Prosessori 9* (2007), s.43 - 45.

[31] Kuittinen, Petri. Sun odottaa paljon JavaFX:ltä. *Prosessori 6-7* (2007), s.20.

[32] Adobe. *System requirements*
[verkkodokumentti] 2008 [viitattu 15.2.2008]. Saatavissa:
http://www.adobe.com/products/air/systemreqs/.

[33] Peltomäki, Juha - Nykänen, Ossi, Web-selainohjelmointi. Jyväskylä: Docendo Finland Oy. 2006.

Application.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<application xmlns="http://ns.adobe.com/air/application/1.0">
  <id>fi.tuomas.StoreBrowser</id>
  <filename>StoreBrowser</filename>
  <name>StoreBrowser</name>
  <version>1.0</version>
  <description/>
  <copyright>Tuomas Haimi 2008</copyright>
  <initialWindow>
    <content>root.html</content>
    <title>Asiakasselain</title>
    <systemChrome>standard</systemChrome>
    <transparent>false</transparent>
    <visible>true</visible>
    <minimizable>true</minimizable>
    <maximizable>true</maximizable>
    <resizable>true</resizable>
    <width>1024</width>
    <height>768</height>
    <x>0</x>
    <y>0</y>
    <minSize>800 600</minSize>
    <maxSize>1280 1024</maxSize>
  </initialWindow>
  <icon>
    <image16x16>icons/AIRApp_16.png</image16x16>
    <image32x32>icons/AIRApp_32.png</image32x32>
    <image48x48>icons/AIRApp_48.png</image48x48>
    <image128x128>icons/AIRApp_128.png</image128x128>
  </icon>
  <customUpdateUI>false</customUpdateUI>
  <allowBrowserInvocation>false</allowBrowserInvocation>
</application>
```

Root.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Asiakasselain</title>
    <script type="text/javascript" src="AIRAliases.js"></script>
    <script type="text/javascript" src="js/root.js"></script>
    <script src="servicemonitor.swf"
      type="application/x-shockwave-flash"/>
    <link rel="stylesheet" href="css/root.css" type="text/css" />
  </head>
  <body>
    <div id="uidiv">
     <iframe id="ui"
       src="ui.html"
       sandboxRoot="http://biltemastorebrowser.fi/"
       documentRoot="app:/ui/"
       allowcrossDomainxhr="true"
       width="100%"
       height="100%"
       scrolling="yes">
     </iframe>
    </div>
  </body>
</html>
<!-- Application init -->
<script>
  var root = new Root();
  root.doLoad();
</script>
```

Root.js

```
/**
 * @author Tuomas Haimi 2007
 */
var configFile;
var configXML;
var stream;
var monitor;
var status;

function Root(){
  this.doLoad = function(){
  this.doFullscreen();
  this.testConnection();
  this.expose();
  this.monitorUserPresence();
}
this.doFullscreen = function(){
  window.nativeWindow.stage.displayState=
    runtime.flash.display.StageDisplayState.FULL_SCREEN_INTERACTIVE;
}
this.monitorUserPresence = function(){
  air.NativeApplication.nativeApplication.idleThreshold = 120;
  air.NativeApplication.nativeApplication.addEventListener
   (air.Event.USER_IDLE, this.onIdle);
  air.NativeApplication.nativeApplication.addEventListener
   (air.Event.USER_PRESENT, this.onPresence);
}
this.onIdle = function(){
  var ui = document.getElementById('ui');
  ui.src = "ui.html";
}
this.onPresence = function(){
  // DUMMY_FUNCTION
}
this.doReload = function(){
  var ui = document.getElementById('ui');
  ui.src = "ui.html";
}
this.testConnection = function(){
  var url = new air.URLRequest('http://localhost/storebrowser/');
  monitor = new air.URLMonitor(url);
  monitor.addEventListener(air.StatusEvent.STATUS,
    this.connectionStatus);
  monitor.pollInterval = 2000;
  monitor.start();
}
this.connectionStatus = function(e){
  if(monitor.available)
    status = 1;
  else
    status = 0;
  return status;
}
```

```
this.expose = function(){
  var Exposed = {};
  Exposed.getConfig = this.getConfig;
  Exposed.doReload = this.doReload;
  Exposed.connectionStatus = this.connectionStatus;
  document.getElementById('ui').
    contentWindow.parentSandboxBridge = Exposed;
}
this.getConfig = function(){
  configFile = air.File.documentsDirectory.resolvePath("settings.xml");
  stream = new air.FileStream();
  if (configFile.exists) {
    stream.open(configFile, air.FileMode.READ);
    configXML = stream.readUTFBytes(stream.bytesAvailable);
    stream.close();
    var domParser = new DOMParser();
    configXML = domParser.parseFromString(configXML, "text/xml");
    var bhrs = configXML.getElementsByTagName("businesshours")[0];
    /* settings-olio */
   var settings = {"serverurl" : configXML.getElementsByTagName
    ("serverurl")[0].firstChild.nodeValue,
     "imageurl" : configXML.getElementsByTagName
    ("imageurl")[0].firstChild.nodeValue,
     "browserid" : configXML.getElementsByTagName
    ("browserid")[0].firstChild.nodeValue,
    "defaultlang" : configXML.getElementsByTagName
    ("defaultlang")[0].firstChild.nodeValue,
    "name" : configXML.getElementsByTagName
    ("name")[0].firstChild.nodeValue,
    "workdays" : bhrs.getAttribute("workdays"),
    "saturday" : bhrs.getAttribute("saturday"),
    "sunday" : bhrs.getAttribute("sunday")
   };
   return settings;
   }
   else{
     return "Can't find settings";
   }
  }
}
```

Ui.html

```
<html>
  <head>
    <meta http-equiv="content-type" content="text-html; charset=utf-8"/>
    <script type="text/javascript" src="js/SpryEffects.js"></script>
    <script type="text/javascript" src="js/effects.js"></script>
    <script type="text/javascript" src="js/ajax.js"></script>
    <script type="text/javascript" src="js/ui.js"></script>
    <script type="text/javascript" src="js/dialog.js"></script>
    <link rel="stylesheet" href="css/ui.css" type="text/css"/>
    title>Biltema Asiakasselain</title>
  </head>
  <body id="uibody" onload="doLoad();">
    <!-- StoreBrowser Navigation Start -->
    <div id="top">
      <img src="css/images/bil3.png" id="logoimg" title="Etusivulle" />
    </div>
    <div id="side">
      <div class="menu_top">
        <div class="menu_heading">Tuotehaku</div>
        <div style="text-align: center; padding: 10px;">
        <form id="search_form" name="search_form" action="javascript:
          getProductList();">
        <input type="hidden" name="action" id="action"
          value="getproducts"></input>
        <table style="width: 100%;">
          <tr>
            <td colspan="2" style="text-align: center;">
              <p class="ohje">
                Voit hakea tuotteita tuotenumerolla tai tuotenimellä.
              </p>
            </td>
          </tr>
          <tr>
            <td>
              <input type="text" name="search_str"
                id="search_str" size="12">
              </input>
            </td>
            <td>
              <img src="css/images/btn_hae_up.png"
                id="search_btn" class="btn"/>
            </td>
          </tr>
        </table>
        </form>
      </div>
    </div>
    <div class="menu_top">
      <div class="menu_heading">Ostoslista</div>
      <div id="plistdiv" style="text-align: center; padding: 10px;">
        <p id="plist">Ostoslista on tyhjä!</p>
        <img src="css/images/btn_show_dsbl.png"
          class="btn_dsbl" id="showl_btn" /> 
        <img src="css/images/btn_print_dsbl.png" id="printl_btn" />
      </div>
    </div>
```

```
    <div class="menu_top">
      <div class="menu_heading">Myymäläinfo</div>
      <div style="text-align: center; padding: 10px;">
        <div id="shopinfo"></div>
          <img src="css/images/btn_palaute_up.png"
            id="feedback_btn" class="btn"/>
        </div>
    </div>
    <div>
      <div id="status">
        <img id="statusimg" src="css/images/pri1.png" />
      </div>
      <div id="ldr" style="display: none;">
        <img src="css/images/loader.gif"/> 
        <img src="css/images/ladataan.jpg"/>
      </div>
    </div>
   </div>
   <!-- StoreBrowser Navigation End -->
   <!-- StoreBrowser Main Content Start -->
   <div id="main">
     <div id="content">
       <div class="menu_heading"> </div>
        <!-- StoreBrowser dynamic content Start -->
        <div id="productlist" class="container"></div>
        <!-- StoreBrowser dynamic content Ends -->
        <div class="clear"></div>
        <div id="sx"></div>
      </div>
     </div>
     <!-- StoreBrowser Main Content End -->
     <!-- Loader Start -->
     <div id="loader" style="display: none;"><br/>
       <img src="css/images/loader.gif"/> 
       <img src="css/images/ladataan.jpg"/><br/> <br/>
     </div>
     <!-- Loader Ends -->
     <!-- Popup Start -->
     <div id="popup" style="display: none;">
       <div id="mtext"></div>
       <div id="ok">
         <img src="css/images/btn_ok_up.png" id="btn_ok" />
       </div>
     </div>
     <!-- Popup Ends -->
  </body>
</html>
```

Ui.js

```
/* @author Tuomas Haimi 2007 */
var settings;
var hide_loader;
var show_window;
var hide_window;
var hlite;
var arrayPageSize;

function doLoad(){
  var logoimg = $('logoimg');
  var search_str = $('search_str');
  var search_btn = $('search_btn');
  var feedback_btn = $('feedback_btn');
  var ok_btn = $('btn_ok');
  var search_form = $('search_form');
  /* add eventlistener to ok-button */
  ok_btn.onclick = function(){
    hidePopup();
  }
  ok_btn.onmouseover = function(){
    this.src = "css/images/btn_ok_down.png";
  }
  ok_btn.onmouseout = function(){
    this.src = "css/images/btn_ok_up.png";
  }
  /* add eventlistener to search_str */
  search_str.onfocus = function(){
    search_str.value = "";
  }
  /* add eventlistener to logoimg */
  logoimg.onclick = function(){
    parentSandboxBridge.doReload();
  }
  /* add eventlistener to search-button */
  search_btn.onclick = function(){
    getProductList();
  }
  search_btn.onmouseover = function(){
    this.src = "css/images/btn_hae_down.png";
  }
  search_btn.onmouseout = function(){
    this.src = "css/images/btn_hae_up.png";
  }
  /* add eventlistener to feedback-button */
  feedback_btn.onclick = function(){
    var _url = settings.serverurl+"?action=feedback";
    var req = new Ajax.httpRequest(_url, showFeedback, true);
  }
  feedback_btn.onmouseover = function(){
    this.src = "css/images/btn_palaute_down.png";
  }
  feedback_btn.onmouseout = function(){
    this.src = "css/images/btn_palaute_up.png";
  }
  /* load config via sandboxbridge */
  settings = parentSandboxBridge.getConfig();
  setShopInfo();
```

```
  hide_loader = new Spry.Effect.Fade("ldr",
    {duration: 2000, from:  '100%', to: '0%', toggle: false});
  clearShoppinglistOnreload();
  var status = parentSandboxBridge.connectionStatus();
  if(status == 1)
    netStatusOk();
  else{
    netStatusNok();
}
function netStatusOk(){
  var statusimg = $('statusimg');
  statusimg.src = "css/images/pri1.png";
}
function netStatusNok(){
  var statusimg = $('statusimg');
  statusimg.src = "css/images/pri3.png";
}
function setShopInfo(){
  var elm = $('shopinfo');
  elm.innerHTML = "<p><b>"+settings.name+"</b></p><p>Myymälä avoinna
   <br />Arkisin: "+settings.workdays+"<br />
   Lauantaisin: "+settings.saturday+"<br />
   Sunnuntaisin: "+settings.sunday+"</p>";
}
function getProductList(){
  var search_str = $('search_str');
  if(search_str.value == ""){
    showPopup("Anna tuotenumero tai tuotenimi!");
  }
  else{
    var _args = "";
    var form = $('search_form');
    var form_length = form.elements.length;
    for(var i = 0; i < form_length; i++){
      if(i < form_length - 1){
        _args += form.elements[i].name+"="+
        encodeURIComponent(form.elements[i].value)+"&";
      }
      else{
        _args += form.elements[i].name+"="+
        encodeURIComponent(form.elements[i].value);
      }
    }
   var _url = settings.serverurl+"?"+_args;
   var req = new Ajax.httpRequest(_url, showProductList);
  }
}
function getProductInfo(productnumber){
  this.productnumber = productnumber;
  var _arg = productnumber.substring(1);
  var _url = settings.serverurl+"?action=
    getproductinfo&productnumber="+_arg;
  var req = new Ajax.httpRequest(_url, showProductInfo);
}
function showProductInfo(){
  var xml = this.req.responseXML;
  initWindow(xml, parseProductXML, productDialog, "Tuotetiedot");
}
```

```
function addToList(productnumber){
  this.productnumber = productnumber;
  var _arg = productnumber.substring(1);
  var _url = settings.serverurl+"?action=addtolist&productnumber="+_arg;
  var req = new Ajax.httpRequest(_url, updateProductlist);
}
function updateProductlist(){
  var showl_btn = $('showl_btn');
  showl_btn.src = "css/images/btn_show_up.png";
  var printl_btn = $('printl_btn');
  printl_btn.src = "css/images/btn_print_up.png";
  var q = this.req.responseText;
  var plist = $('plist');
  if(q == 1)
    plist.innerHTML = "Ostoslistassa on "+q+" tuote.";
  else
   plist.innerHTML = "Ostoslistassa on "+q+" tuotetta.";
  Spry.Effect.DoHighlight('plistdiv',
    {duration:1000,from:'#ffff99',to:'#eff5ff'});
  /* add eventlistener to showl-button */
  showl_btn.onclick = function(){
    getShoppinglist();
  }
  showl_btn.onmouseover = function(){
    this.className = "btn_over";
    this.src = "css/images/btn_show_down.png";
  }
  showl_btn.onmouseout = function(){
    this.className = "btn";
    this.src = "css/images/btn_show_up.png";
   }
   /* add eventlistener to printl-button */
   printl_btn.onclick = function(){
     clearShoppinglist();
   }
   printl_btn.onmouseover = function(){
     this.className = "btn_over";
     this.src = "css/images/btn_print_down.png";
   }
   printl_btn.onmouseout = function(){
     this.className = "btn";
     this.src = "css/images/btn_print_up.png";
   }
   showl_btn.className = "btn";
   printl_btn.className = "btn";
}
function clearShoppinglistOnreload(){
   var _url = settings.serverurl+"?action=clearlist";
   var req = new Ajax.httpRequest(_url, DUMMY_USE, false);
}

function clearShoppinglist(){
  var _url = settings.serverurl+"?action=clearlist";
  var req = new Ajax.httpRequest(_url, DUMMY_USE);
  var showl_btn = $('showl_btn');
  showl_btn.src = "css/images/btn_show_dsbl.png";
  var printl_btn = $('printl_btn');
  printl_btn.src = "css/images/btn_print_dsbl.png";
  printl_btn.onclick = function(){}
  printl_btn.onmouseover = function(){}
  printl_btn.onmouseout = function(){}
```

```
  showl_btn.onclick = function(){}
  showl_btn.onmouseover = function(){}
  showl_btn.onmouseout = function(){}
  $('plistdiv').style.bgColor = 'transparent';
  showl_btn.className = "btn_dsbl";
  printl_btn.className = "btn_dsbl";
  $('plist').innerHTML = "Ostoslista on tyhjä!";
}
function getShoppinglist(){
  var _url = settings.serverurl+"?action=showlist";
  var req = new Ajax.httpRequest(_url, showShoppinglist);
}
function showShoppinglist(){
  var xml = this.req.responseXML;
  initWindow(xml, parseXMLtoHTML, shoppinglistDialog, "Ostoslista");
}
function showProductList(){
  var div = $('productlist');
  div.style.display = 'none';
  var html = parseProductXml(this.req.responseXML);
  div.innerHTML = html;
  show_window = new Spry.Effect.Slide("productlist",
   {duration: 300, from: '0%', to: '100%', toggle: false});
  show_window.start();
  var trs = document.getElementsByClassName('troff');
  var show_btn = document.getElementsByClassName('btn showbtn');
  var add_btn = document.getElementsByClassName('btn addbtn');
  // add eventlistener to table tr's
  for (var i = 0; i < trs.length; i++){
   $(trs[i].id).onmouseover = function(){
     this.className = "tron";
   }
   $(trs[i].id).onmouseout = function(){
     this.className = "troff";
    }
  }
  // add eventlistener to show-buttons
  for (var i = 0; i < show_btn.length; i++){
    $(show_btn[i].id).onclick = function() {
      getProductInfo(this.id);
    }
    $(show_btn[i].id).onmouseover = function(){
      this.src = "css/images/btn_show_down.png";
    }
    $(show_btn[i].id).onmouseout = function(){
      this.src = "css/images/btn_show_up.png";
    }
   }
   // add eventlistener to add-buttons
   for (var i = 0; i < add_btn.length; i++){
      $(add_btn[i].id).onclick = function(){
        addToList(this.id);
      }
      $(add_btn[i].id).onmouseover = function(){
        this.src = "css/images/btn_add_down.png";
      }
      $(add_btn[i].id).onmouseout = function(){
        this.src = "css/images/btn_add_up.png";
      }
    }
```

```
  this.req = null;
  show_window = new Spry.Effect.Slide("productlist",
   {duration: 300, from: '0%', to: '100%', toggle: false});
  show_window.start();
}
function parseProductXml(xmldoc){
  var number = xmldoc.getElementsByTagName('productnumber');
  var name = xmldoc.getElementsByTagName('productname');
  var price = xmldoc.getElementsByTagName('productprice');
  var saldo = xmldoc.getElementsByTagName('productsaldo');
  var place = xmldoc.getElementsByTagName('productplace');
  var html;
  if(number.length == 0)
    html = "<h3>Hakemaasi tuotetta ei löytynyt</h3>"
  else{
    html = "<div class=\"test\"><table class=\"productlist_tbl\"><tr>
            <td class=\"dg_header\">Tuotenumero</td>
            <td class=\"dg_header\">Tuotenimi</td>
            <td class=\"dg_header\">Hinta</td>
            <td class=\"dg_header\">Saldo</td>
            <td class=\"dg_header\">Varastopaikka</td>
            <td class=\"dg_header\"></td>
            <td class=\"dg_header\"></td></tr>";
    for(var i = 0; i < number.length; i++){
      html += "<tr class=\"troff\" id=\"tr"+i+"\" >";
      html += "<td>"+number[i].firstChild.nodeValue+"</td>
               <td>"+name[i].firstChild.nodeValue+"</td>";
      html += "<td>"+price[i].firstChild.nodeValue+"</td>
               <td>"+saldo[i].firstChild.nodeValue+"</td>
               <td>"+place[i].firstChild.nodeValue+"</td>";
      html += "<td><img src=\"css/images/btn_show_up.png\" class=\"btn
                    showbtn\" id=\"g"+number[i].
                    firstChild.nodeValue+"\" /></td>";
      html += "<td><img src=\"css/images/btn_add_up.png\" class=\"btn
                    addbtn\" id=\"a"+number[i].
                    firstChild.nodeValue+"\" /></td>";
      html += "</tr>";
    }
    html += "</table></div>";
   }
   return html;
}
function showFeedback(){
  var html = this.req.responseText;
  initWindow(html, buildFeedbackForm, feedbackDialog, "Asiakaspalaute")
}
function sendFeedback(){
  var _args = "";
  var form = $('feedback_form');
  var form_length = form.elements.length;
  for(var i = 0; i < form_length; i++){
    if(i < form_length - 1){
      _args += form.elements[i].name+"="+
      encodeURIComponent(form.elements[i].value)+"&";
    }
    else{
     _args += form.elements[i].name+"="+
     encodeURIComponent(form.elements[i].value);
    }
  }
  alert(_args);
```

```
  var _url = settings.serverurl+"?"+_args;
  var req = new Ajax.httpRequest(_url, handleFeedback);
}
function handleFeedback(){
  var result = this.req.responseText;
  if(result == 1){
   show_window = new Spry.Effect.Fade("shadow",
    {duration: 300, from: '100%', to: '0%', toggle:
    false, finish: hideOverlay});
   show_window.start();
  }
}
function showLoader(){
  var arrayPageSize = getPageSize();
  var loader = $('ldr');
  loader.style.display = 'block';
}
function hideLoader(){
  hide_loader.start();
}
function showPopup(message){
  hide_popup = new Spry.Effect.Fade("popup",
   {duration: 400, from: '0%', to: '100%', toggle: false});
  hide_popup.start();
  popup.style.top = 200 +'px';
  popup.style.left = 40 +'px';
  var mtext = $('mtext');
  mtext.innerHTML = "<h3>"+message+"</h3>";
}
function hidePopup(){
  popup.style.display = 'none';
}

/********** TOOL FUNCTIONS **************/

function Trim(str){
  while(str.charAt(0) == (" ") ){
    str = str.substring(1);
  }
  while(str.charAt(str.length-1) == " " ){
    str = str.substring(0,str.length-1);
  }
  return str;
}
function $(){
  var element = arguments[0];
  if (typeof element == 'string')
     element = document.getElementById(element);
  return element;
}
/***
  The Javascript DOM From Stephen Chapman,
  http://javascript.about.com/library/bldom08.htm
*/
document.getElementsByClassName = function(cl) {
  var retnode = [];
  var myclass = new RegExp('\\b'+cl+'\\b');
  var elem = this.getElementsByTagName('*');
```

```
  for (var i = 0; i < elem.length; i++) {
    var classes = elem[i].className;
    if (myclass.test(classes)) retnode.push(elem[i]);
  }
  return retnode;
}
function getPageScroll(){
  var yScroll;
  if (self.pageYOffset) {
    yScroll = self.pageYOffset;
  }
  else if (document.documentElement &&
    document.documentElement.scrollTop){     // Explorer 6 Strict
      yScroll = document.documentElement.scrollTop;
  }
  else if (document.body) {// all other Explorers
    yScroll = document.body.scrollTop;
  }
  arrayPageScroll = new Array('',yScroll)
  return arrayPageScroll;
}
function getPageSize(){
  var xScroll, yScroll;
  if (window.innerHeight && window.scrollMaxY) {
    xScroll = document.body.scrollWidth;
    yScroll = window.innerHeight + window.scrollMaxY;
  }
  else if (document.body.scrollHeight > document.body.offsetHeight){
    xScroll = document.body.scrollWidth;
    yScroll = document.body.scrollHeight;
  }
  else {
    xScroll = document.body.offsetWidth;
    yScroll = document.body.offsetHeight;
  }
  var windowWidth, windowHeight;
  if (self.innerHeight) {
    windowWidth = self.innerWidth;
    windowHeight = self.innerHeight;
  }
  else if (document.documentElement &&
    document.documentElement.clientHeight) {
    windowWidth = document.documentElement.clientWidth;
    windowHeight = document.documentElement.clientHeight;
  }
  else if (document.body) {
    windowWidth = document.body.clientWidth;
    windowHeight = document.body.clientHeight;
  }
  if(yScroll < windowHeight)
   pageHeight = windowHeight;
  else
   pageHeight = yScroll;
  if(xScroll < windowWidth)
   pageWidth = windowWidth;
  else
   pageWidth = xScroll;
  arrayPageSize = new Array
   (pageWidth,pageHeight,windowWidth,windowHeight)
  return arrayPageSize;
}
```

Ajax.js

```
/**
 * @author Tuomas Haimi 2007
 */

/* namespacing object */
var Ajax = new Object();

Ajax.RS_UNINITIALIZED = 0;
Ajax.RS_LOADING = 1;
Ajax.RS_LOADED = 2;
Ajax.RS_INTERACTIVE = 3;
Ajax.RS_COMPLETE = 4;

Ajax.httpRequest = function
  (url,onload,onerror,method,params,contentType){
  var status = parentSandboxBridge.connectionStatus();
  if(status == 1){
    this.req=null;
    this.onload=onload;
    this.onerror=(onerror) ? onerror : this.defaultError;
    this.loadXMLDoc(url,method,params,contentType);
    showLoader();
    netStatusOk();
  }
  else{
    netStatusNok();
    /*alert("Ei verkkoyhteyttä!")*/
  }
}

Ajax.httpRequest.prototype.loadXMLDoc = function
  (url,method,params,contentType){
   if (!method){
     method = "GET";
   }
   if (!contentType && method == "POST"){
     contentType = 'application/x-www-form-urlencoded';
   }
   this.req = new XMLHttpRequest();
   if (this.req){
     try{
         var loader = this;
         this.req.onreadystatechange = function(){
           Ajax.httpRequest.onReadyState.call(loader);
         }
         this.req.open(method,url,true);
         if (contentType){
           this.req.setRequestHeader('Content-Type', contentType);
         }
         this.req.send(params);
     }
     catch (err){
       alert(err);
       this.onerror.call(this);
     }
   }
 }
```

```
Ajax.httpRequest.onReadyState = function(){
  var req = this.req;
  var ready = req.readyState;
  if (ready == Ajax.RS_COMPLETE){
    var httpStatus = req.status;
    if (httpStatus == 200 || httpStatus == 0){
      hideLoader();
      this.onload.call(this);
    }
    else{
      hideLoader();
      this.onerror.call(this);
    }
  }
}

Ajax.httpRequest.prototype.defaultError = function(){
  alert("error fetching data!"
    +"\n\nreadyState:"+this.req.readyState
    +"\nstatus: "+this.req.status
    +"\nheaders: "+this.req.getAllResponseHeaders());
}

function DUMMY_USE(){

}
```

Dialog.js

```
/* @author Tuomas Haimi 2007 */

function initOverlay(){
  var elmBody = document.getElementsByTagName("body").item(0);
  var elmOverlay = document.createElement("div");
  elmOverlay.setAttribute('id','overlay');
  elmOverlay.style.display = 'none';
  elmBody.appendChild(elmOverlay);
}
function showOverlay(onReady){
  this.onReady = onReady;
  var arrayPageSize = getPageSize();
  var elmOverlay = document.getElementById('overlay');
  elmOverlay.style.height = arrayPageSize[1] +"px";
  var show_overlay = new Spry.Effect.Fade("overlay",
    {duration: 500, from: '0%', to: '100%', toggle: false,
    finish: onReady});
  show_overlay.start();
}
function hideOverlay(){
  var elmOverlay = document.getElementById('overlay');
  var box = document.getElementById('shadow');
  elmOverlay.removeChild(box);
  var hide_overlay = new Spry.Effect.Fade("overlay",
   {duration: 300, from: '100%', to: '0%', toggle: false,
   finish: removeOverlay});
  hide_overlay.start();
}
function removeOverlay(){
  var elmBody = document.getElementsByTagName("body").item(0);
  var elmOverlay = document.getElementById('overlay');
  elmBody.removeChild(elmOverlay);
}

/*********************************************************/

var _htext = "Tuote ikkuna";

function initWindow(xml, parser, onReady, text){
  this.xml = xml;
  this.parser = parser;
  this.onReady = onReady;
  this.text = text;
  var html = parser(xml);
  initOverlay();
  var body = document.getElementById('overlay');
  var shadow = document.createElement("div");
  shadow.setAttribute('id','shadow');
  var box = document.createElement("div");
  box.setAttribute('id','window_box');
  var header = document.createElement("div");
  header.setAttribute('id','window_header');
  var cnt = document.createElement("div");
  cnt.setAttribute('id','window_content');
  cnt.innerHTML = html;
```

```
  shadow.appendChild(box);
  body.appendChild(shadow);
  box.appendChild(header);
  box.appendChild(cnt);
  var htxt = document.createTextNode(text);
  header.appendChild(htxt);
  shadow.style.left = (arrayPageSize[2]-600)/2+'px';
  shadow.style.top = '200px';
  showOverlay(onReady);
}
function productDialog(){
  show_window = new Spry.Effect.Slide("window_content",
    {duration: 300, from: '0%', to: '100%', toggle: false});
  show_window.start();
  var closebtn = $('closebutton');
  closebtn.onmouseover = function(){
    this.className = "btn_over";
    this.src = "css/images/btn_sulje_down.png";
  }
  closebtn.onmouseout = function(){
    this.className = "btn";
    this.src = "css/images/btn_sulje_up.png";
  }
  closebtn.onclick = function(){
    show_window = new Spry.Effect.Fade("shadow",
      {duration: 300, from: '100%', to: '0%', toggle: false,
      finish: hideOverlay});
    show_window.start();
  }
}
function shoppinglistDialog(xml){
  show_window = new Spry.Effect.Slide("window_content",
    {duration: 300, from: '0%', to: '100%', toggle: false});
  show_window.start();
  var closebtn = $('closebutton');
  closebtn.onmouseover = function(){
    this.className = "btn_over";
    this.src = "css/images/btn_sulje_down.png";
  }
  closebtn.onmouseout = function(){
    this.className = "btn";
    this.src = "css/images/btn_sulje_up.png";
  }
  closebtn.onclick = function(){
    show_window = new Spry.Effect.Fade("shadow",
      {duration: 300, from: '100%', to: '0%', toggle: false,
      finish: hideOverlay});
    show_window.start();
  }
}
function feedbackDialog(){
  show_window = new Spry.Effect.Slide("window_content",
    {duration: 300, from: '0%', to: '100%', toggle: false});
  show_window.start();
  var sendbtn = $('btn_send');
  var closebtn = $('btn_close');



  sendbtn.onmouseover = function(){
    this.className = "btn_over";
```

```
    this.src = "css/images/btn_send_down.png";
  }
  sendbtn.onmouseout = function(){
    this.className = "btn";
    this.src = "css/images/btn_send_up.png";
  }
  sendbtn.onclick = function(){
    sendFeedback();
   }
  closebtn.onmouseover = function(){
    this.className = "btn_over";
    this.src = "css/images/btn_sulje_down.png";
  }
  closebtn.onmouseout = function(){
    this.className = "btn";
    this.src = "css/images/btn_sulje_up.png";
  }
  closebtn.onclick = function(){
    show_window = new Spry.Effect.Fade("shadow",
     {duration: 300, from: '100%', to: '0%', toggle: false,
     finish: hideOverlay});
    show_window.start();
  }
}
function buildFeedbackForm(html){
  this.html = html;
  return html;
}
function parseXMLtoHTML(xml){
  this.xml = xml;
  var html;
  var price;
  var sum = 0;
  var product_number =
    xml.getElementsByTagName('productnumber');
  var product_name = xml.getElementsByTagName('productname');
  var product_price =
    xml.getElementsByTagName('productprice');
  var product_place =
    xml.getElementsByTagName('productplace');
  html = "<div class=\"xx\"><table class=\"sltbl\">";
  html += "<tr>";
  html += "<td class=\"slhd_l\">Tuotenumero</td>
           <td class=\"slhd_c\">Tuotenimi</td>
           <td class=\"slhd_c\">Hinta</td>
           <td class=\"slhd_r\">Hylly</td>";
  html += "</tr>";
  for(var i = 0; i < product_number.length; i++){
    if(i % 2){
      html += "<tr class=\"trl\">";
    }
    else{
     html += "<tr>";
    }
    html += "<td>"+product_number[i].
      firstChild.nodeValue+"</td>";
   html += "<td>"+product_name[i].
     firstChild.nodeValue+"</td>";
   html += "<td>"+product_price[i].
     firstChild.nodeValue+"</td>";
```

```
   html += "<td>"+product_place[i].
     firstChild.nodeValue+"</td>";
   html += "</tr>";
   sum = sum + parseFloat(product_price[i].
    firstChild.nodeValue);
  }
  html += "<tr><td class=\"tdv\" colspan=\"4\"></td></tr>";
  html += "<tr><td colspan=\"2\">Yhteensä:</td>
              <td>"+sum.toFixed(2)+"</td><td></td></tr>";
  html += "<tr><td class=\"center\" colspan=\"4\"><br />
    <img src=\"css/images/btn_sulje_up.png\"
    class=\"btn\" id=\"closebutton\" /></td></tr>";
  html += "</table><div>";
 return html;
}
function parseProductXML(xml){
  this.xml = xml;
  var html;
  var product_number = xml.getElementsByTagName('tuote_nro');
  var length = (product_number[0].
    firstChild.nodeValue).length;
  var imglink = (product_number[0].firstChild.nodeValue).
    slice(0,2)+"_"+(product_number[0].firstChild.nodeValue).
    slice(2,length)+"iso.jpg";
  var product_name = xml.getElementsByTagName('tuote_nimi');
  var product_text = xml.getElementsByTagName('tuote_teksti');
  var product_price = xml.getElementsByTagName('tuote_hinta');
  var product_saldo = xml.getElementsByTagName('tuote_saldo');
  var product_place = xml.getElementsByTagName('tuote_vp');
  html = "<table class=\"product_tbl\"><tr>";
  html += "<td class=\"product_td\">
    <div class=\"product_img\">
    <img src=\""+settings.imageurl+imglink+"\" /></div></td>";
  html += "<td class=\"product_td\">
    <h3>" + product_name[0].firstChild.nodeValue + "</h3>";
  html += "<br /><p>" + product_text[0].firstChild.nodeValue +
    "</p><p><b>Tuotenumero: <span id=\"prodn\">" +
    product_number[0].firstChild.nodeValue + "</span>";
  html += "<br />Saldo: " +
    product_saldo[0].firstChild.nodeValue + "";
  html += "<br />Hylly: " +
    product_place[0].firstChild.nodeValue +
    "</b></p></td></tr>";
  html += "<tr><td class=\"center\" colspan=\"4\">
   <img src=\"css/images/btn_sulje_up.png\"
   class=\"btn\" id=\"closebutton\" /></td></tr>";
  html += "</table><br /><br />";
  return html;
}
```